Universidad Rey Juan Carlos

TESIS DOCTORAL

A new methodology for the automated generation of reproducible metaheuristic configurations: a practical application to combinatorial optimization problems

Autor:

D. Raúl Martín Santamaría

Directores:

Dr. D. José Manuel Colmenar Verdugo

Dr. D. Abraham Duarte Muñoz

Programa de doctorado en Tecnologías de la Información y las Comunicaciones

Escuela Internacional de Doctorado

2023

This doctoral thesis has been supported by the following projects and grants:

- Project "NUEVA METODOLOGIA HOLISTICA PARA LA CONFIGURACION, COMPARACION Y EVALUACION DE METAHEURISTICAS", grant ref PID2021-126605NB-I00 funded by MCIN/AEI/10.13039/501100011033, and ERDF A way of making Europe.
- Comunidad de Madrid y Fondos Estructurales de la Unión Europea with grant ref. P2018/TCS-4566.
- Project "EFFICIENT METAHEURISTICS IN GRAPH OPTIMIZATION", funded by Spanish Ministerio de Ciencia, Innovación y Universidades under grant ref. PGC2018-095322-B-C22.

MINISTERIO
DE CIENCIA
E INNOVACIÓN

UNIÓN EUROPEA
FONDO EUROPEO DE
DESARROLLO REGIONAL
"Una manera de hacer Europa"

AGENCIA
ESTATAL DE
INVESTIGACIÓN

El Dr. D. Abraham Duarte Muñoz, Profesor Catedrático de Universidad del Departamento de Informática y Estadística de la Universidad Rey Juan Carlos, y el Dr. D. José Manuel Colmenar Verdugo, Profesor Titular de Universidad del Departamento de Informática y Estadística de la Universidad Rey Juan Carlos, directores de la Tesis titulada: "A new methodology for the automated generation of reproducible metaheuristic configurations: a practical application to combinatorial optimization problems" realizada por el doctorando D. Raúl Martín Santamaría,

HACEN CONSTAR:

que esta Tesis Doctoral reúne los requisitos necesarios para su defensa y aprobación.

En Móstoles, a fecha de la firma electrónica,

Dr. D. Abraham Duarte Muñoz Dr. D. José Manuel Colmenar Verdugo

FIRMADO POR	FECHA FIRMA
COLMENAR VERDUGO JOSE MANUEL	28-03-2023 09:27:27
DUARTE MUÑOZ ABRAHAM - DIRECTOR DE LA ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA	28-03-2023 14:31:38

Acknowledgements

Aquí estamos. Hemos llegado. Había luz al final del túnel. Años de trabajo culminan con el presente manuscrito.

Gracias a mi familia, por su apoyo incondicional. Sin ellos, no tendría la motivación ni la confianza para perseguir mis objetivos. Sus ánimos y confianza en mí ha sido una de las mayores fuerzas a lo largo de todo el doctorado.

A todos mis compañeros, por el tiempo compartido tanto en el departamental como fuera de él. Es incalculable el valor que tiene haber tenido la oportunidad de trabajar con un grupo de personas con tanto talento y motivación. Gracias por escucharme, mis dudas, mis miedos, y aguantar alguna que otra chapa.

A mis directores de tesis, por introducirme al campo de la optimización. Gracias por la confianza depositada en el increíble proyecto en el que nos hemos embarcado. Sin su apoyo desinteresado y el tiempo que han invertido en mi formación no hubiera llegado hasta aquí.

A Thomas, por acogernos a Sergio y a mí en Bruselas durante el verano. Gracias por todo el apoyo prestado durante la estancia en la *Université Libre de Bruxelles*. Gracias también a todos los miembros de *Iridia*, que nos acogieron allí como si nos conocieran de toda la vida. Conocí una ciudad increíble, y a mucha gente que no olvidaré.

A todos los compañeros del departamental, por vuestra experiencia y consejos. He aprendido un montón de todos vosotros, especialmente de los puntos de vista más antagónicos, que me han forzado a ver las cosas de una manera diferente.

Gracias por supuesto a todos los docentes que he tenido a lo largo de mi educación, especialmente en el instituto, por alentar mi pasión por la tecnología y las matemáticas. Sin ellos dudo que hubiera elegido la trayectoria que he realizado.

Por último, pero no por ello menos importante, gracias a todas las personas que han compartido mi camino, y han contribuido su granito de arena. No podría haber llegado a donde estoy sin todos vosotros, y por eso, os estaré eternamente agradecido.

A todos vosotros, gracias.

The Weight of Knowledge

In the world of academia,
    Where knowledge is the currency,
There are those who toil and grind,
    For their PhDs, a one-of-a-kind.

They work hard, day and night,
    To earn their place in the ivory tower,
But their struggles are not small,
    For they have little money to empower.

They toil away, without rest,
    For their research, their passion, their quest,
But the funds are scarce, the bills are high,
    And the weight of it all can make them sigh.

They sacrifice so much for their dreams,
    But the path to success is not what it seems,
For even with a PhD in hand,
    Their future is not always grand.

They struggle to find a job,
    To make ends meet, to afford their own sod,
But they persevere, they fight on,
    For their love of learning, their thirst for knowledge, their passion.

So let us not forget the sacrifice,
    Of those who seek to enlighten and advise,
For they are the future of our world,
    And their struggles should not go unfurled.

ChatGPT, 2023

Contents

List of Figures

List of Tables

List of Acronyms

AAC	Automatic Algorithm Configuration.
ACO	Ant Colony Optimization.
ALNS	Adaptive Large Neighborhood Search.
API	Application Programming Interface.
AUC	Area Under Curve.
BMSSC	Balanced Minimum Sum-of-Squares Clustering.
BVNS	Basic Variable Neighborhood Search.
CAP	Corridor Allocation Problem.
CL	Candidate List.
CVRP	Capacitated Vehicle Routing Problem.
DBI	Davies-Bouldin index.
DRFLP	Double Row Facility Layout problem.
FIFO	First In First Out.
FLP	Facility Layout Problem.
FMS	Flexible Manufacturing Systems.
GA	Genetic Algorithm.
GRASP	Greedy Randomized Adaptive Search Procedure.
GVNS	General Variable Neighborhood Search.
HCOT	Hierarchical Clustering with Optimal Transport.
IG	Iterated Greedy.

ILS Iterated Local Search.

JCR Journal Citation Reports.

KPROP K-Parallel Row Ordering Problem.

LIFO Last In First Out.

LNCS Lecture Notes in Computer Science.

MIP Mixed-Integer Programming.

ML Machine Learning.

MRFLP Multi-Row Facility Layout Problem.

MSE Mean Squared Error.

MSSC Minimum Sum-of-Squares Clustering.

PCA Principal Component Analysis.

PR Path Relinking.

PSO Particle Swarm Optimization.

RCL Restricted Candidate List.

SF-DRFLP Space-Free Double Row Facility Layout Problem.

SJR Scimago Journal & Country Rank.

SO Strategic Oscillation.

SRFLP Single Row Facility Layout Problem.

TS Tabu Search.

TSP Traveling Salesman Problem.

TTTPlot Time-to-target Plot.

VND Variable Neighborhood Descent.

VNS Variable Neighborhood Search.

VRP Vehicle Routing Problem.

VRPOD Vehicle Routing Problem with Ocassional Drivers.

VRPTW Vehicle Routing Problem with Time Windows.

Abstract

"You cannot make progress without making decisions". - Jim Rohn

Every day, we are bombarded with decisions: how to travel to a specific destination; which foods will make our meal; how to best organize our closets. Optimization problems are everywhere: engineering, logistics, biology, economy... and of course, in our day-to-day lives. All optimization problems have something in common: we want to reach a certain set of objectives, according to a set of restrictions.

Optimization problems can be commonly solved using two distinct techniques: exact methods, and approximate methods. Exact methods are able to find the best existing solutions, but when applied to most real-life optimization problems, they scale poorly, and require enormous computing resources and large execution time with modest problem sizes. On the other hand, approximate methods, such as heuristic and metaheuristic algorithms, can find good quality solutions using few resources, but they cannot know if there are better solutions to the solutions they find, or if on the contrary any generated solution is optimal.

While metaheuristic algorithms have become one of the most popular methods for solving optimization problems, some issues have been highlighted in the literature. Specifically, two of the most common issues are lack of both reproducibility and reusability of the approaches; and *adhoc* decisions, based on the researcher's experience, that may be difficult to justify from a purely scientific point of view.

To this end, in this doctoral thesis, a new methodology for the automated generation of reproducible metaheuristic configurations is presented. The proposal will not only be theoretical, a reference implementation, called *Mork* (Metaheuristic Optimization frameworRK) will be provided and tested. The benefits of the methodology and its corresponding implementation will be demonstrated against three completely different optimization problems, belonging to unrelated problem families: a facility layout problem, a vehicle routing problem and a clustering problem.

Part I

PhD Dissertation

Contents

Chapter 1

Introduction

There is a growing concern over the lack of methodological frameworks to compare stochastic algorithms in the research community. Instance selection, experimental design and artifacts availability are some examples of the most common problems [2, 3]. In this chapter, we will introduce and detail their importance, summarizing relevant terminology and existing proposals.

1.1 Optimization problems

In our everyday lives, we are continuously making decisions. We decide how much gas to put in our car each time we refuel; which items to pick from the menu when eating out; in which order our daily tasks are going to be solved; and so on.

In the Mathematics and Computer Science fields, these problems are known as optimization problems. Given an objective, or a set of objectives, where we usually have to maximize benefits, or minimize costs, and under a set of problem specific restrictions, our goal is to find good solutions from all possible combinations.

Optimization problems can be classified into continuous and combinatorial, according to their type of variables. Deciding how much fuel to put in your car would be a continuous problem, as the fuel quantity is an arbitrary number, only restricted by the car capacity. However, deciding which car to buy is combinatorial, as the number of values that we can choose from is a discrete number, the number of available options.

According to the complexity of the optimization problem, they can be classified into different complexity classes. If the optimal solution can be found in polynomial time, it is said that the problem belongs to class P. Likewise, if no such algorithm exists, we say that the problem is NP [4]. Although it is currently unknown if $P = NP$, it is widely believed that it is not the case [5,6], and we will work under the assumption that $P \neq NP$, or in other words, there are problems for which the optimal solution can not be found in polynomial time.

There are mainly two types of techniques to solve optimization problems, exact methods and approximate methods. Exact methods guarantee the optimality of the solution if found, but usually fail or are extremely slow as the problem size increases on NP problems. Examples of classical exact approaches are the Simplex method [7], and branch and cut approaches [8]. On the other hand, approximate methods do not guarantee the optimal solution, but are faster, and can tackle considerably larger problems. Examples of approximate methods are heuristic and metaheuristic methods [9,10].

Although the ideas contained in this thesis may be applicable to more approaches or techniques, in this thesis, we will focus exclusively on developing and validating approaches in the context of stochastic algorithms, which includes metaheuristic approaches.

1.2 Algorithm tuning

Heuristic and metaheuristic approaches usually require a parameter tuning process in order to make the proposed algorithm robust, but also flexible, and, in general, to improve the effectiveness and efficiency of the method [11]. Finding the optimal components and parameters for an algorithm can be considered itself as a hard continuous optimization problem, where the performance of the method is the objective function, and the rules that define invalid configurations are the restrictions. Due to its hardness and importance, there are many different approaches used by researchers.

Algorithm configuration can be classified in two types according to their scope: structural and parametric [11]. The first one is concerned with how algorithm components are instantiated and their composition, while the second one decides the value for the algorithm parameters.

The most extended parameter tuning strategy is manually setting parameter values [12]. This strategy can be summarized as follows: for each configurable parameter, execute a preliminary experiment, in which all parameters are configured using values selected according to the researcher experience, changing only the value of the parameter being tested. The value for which the parameter being tested obtains the best value is fixed, and the next experiments use this previous value for that parameter. This strategy is then repeated iteratively for all parameters.

Variants of this strategy can also be applied in an additive fashion. For instance, constructive methods could be tested first in one experiment, and then execute another experiment in which the best constructive found is executed with all local search methods. One component at a time is added and tuned in each successive experiment until the algorithm proposal is considered complete.

Although manual testing may be effective when adjusting only a few parameters, and obtains better results than no parameter tuning at all, it has several important drawbacks. First, due to the dependence on the researcher's experience and intuition, experiments are biased, and their conclusions are not always reproducible by other researchers. Second, it is a highly time-consuming task, and due to the high human intervention required, prone to errors. Third, parameter interactions are rarely tested [13]. For example, in a simple algorithm formed by a constructive method and a local search, a suboptimal constructive configuration may obtain better results when followed by a certain local search than the best constructive found during the preliminary experimentation. If cross effects between parameters are not measured, promising configurations will be discarded due to bad performance in a particular experiment, whereas they may perform well under a different experiment.

Conversely, Automatic Algorithm Configuration (AAC) strategies try to find good algorithm parameter configurations with little human intervention, observing the results obtained using a training or preliminary set of instances [11]. AAC tools are powerful optimizers for mixed-integer stochastic black-box problems. The AAC problem is mixed-integer because algorithmic parameters are often categorical and numerical, black-box because there is no explicit mathematical model of the behavior of the algorithm being configured (target algorithm) and the only way to evaluate the quality of a potential algorithmic configuration is to execute it on a particular problem instance, and stochastic because the target algorithm is often stochastic, as in the case of most metaheuristics. Some of the best known techniques are based on fractional experiment designs [14], racing approaches [15], heuristic searches [16, 17], and statistical modeling [18].

One of the most successful AAC strategies applied to parametric tuning is based on racing and Friedman's analysis of variance by ranks [19]. This proposal, usually denoted as F-Race, was later improved by sampling configurations from the parameter space and refining the sampling distribution by means of repeated applications of F-Race. The resulting automatic configuration approach, called iterated F-race (I/F-Race), is thoroughly described in [20]. More recently, a new strategy called `irace` was introduced [21], where I/F-Race is a special case. It also includes advanced strategies such us the use of the paired test instead of Friedman's test, sampling from truncated normal distribution, parallel implementation, a restart strategy that avoids premature convergence, and an elitist racing procedure to ensure that the best parameter configurations found are also evaluated on the highest number of instances.

While in depth studies of parametric algorithm tuning have been conducted, structural tuning has received less attention. See Chapter 9 for a more detailed algorithm tuning state of the art.

Regarding instance selection for algorithm tuning, there are problematic decisions that researchers may take when choosing a preliminary experimentation set. Two strategies that researchers may be tempted to make, but should be avoided are: using the full instance dataset when tuning an approach, and cherry-picking certain instances due to their properties. If the tuning or preliminary instance set has been chosen by either strategy, the generated algorithm configurations may behave poorly against previously unseen datasets [22]. This phenomena is akin to overfitting in a ML context.

1.3 Hypothesis and objectives

As previously introduced in Section 1.2, there is a knowledge gap in the research community about benchmark instance selection and automatic configuration of algorithms, specially when taking reproducibility factors into account. Therefore, our main hypothesis is that by proposing a holistic methodology that allows researchers to automatically generate metaheuristic approaches, the robustness, quality and reproducibility of metaheuristic approaches can be improved, whilst minimizing the effort to develop them.

Therefore, the main objective of this PhD work is to develop and validate a new methodology based on scientific support for the application of metaheuristic methods in optimization problems. In particular, the specific objectives are the following:

1. Propose a methodological approach to increase the reproducibility of empirical results. We will specifically focus on two key points: automatic selection of test instances, according to their characteristics, and automated generation and validation of algorithms.

2. Develop and publish a set of open-source software tools publicly accessible to the whole scientific community, implementing and automating the proposed methodology.

3. Validate the proposed methodology with well-known combinatorial optimization problems of different unrelated families. This objective includes publishing all artifacts (source code, instances, executable artifacts, and processed results) in publicly available repositories.

4. Publish all results in relevant journals and both national and international conferences.

1.4 Memory structure

In this first chapter, the context in which this thesis is developed has been introduced, and the starting hypothesis and concrete objectives have been defined.

In Chapter 2, the methodological framework is presented. Specifically, both general guidelines, as well as an example framework implementation, will be provided, tackling the concerns presented in Chapter 1.

Next, in Chapter 3, an introduction to heuristic and metaheuristic methods is presented. Then, three relevant optimization problems from completely different families, including their context, practical applications, mathematical formulation and state of the art are summarized.

Subsequently, in Chapter 4, the proposed methodological framework and its implementation is demonstrated against the problems presented in Chapter 3 along with the artifacts generated and their respective references.

Finally, conclusions and future lines of research are drawn in Chapter 5. Immediately after, all achieved contributions during the thesis development will be listed, including journal articles and both international and national conferences presentations. In addition, a short summary of the thesis in Spanish is provided.

Chapter 2

Proposed automated methodology

The goal of this chapter is to detail the methodological framework to promote reproducibility efforts in optimization problems being approached with stochastic algorithms. To do so, an empirical methodology based on minimizing the number of decisions taken by researchers (i.e., automating the majority of them) is presented. The proposed methodology is agnostic to both the concrete metaheuristic algorithms and the programming languages used to implement them.

2.1 Overview

In this section, the main parts of the proposed methodology are summarized. Our main objective is encouraging and favoring reproducibility efforts for optimization problems being solved with stochastic methods, and, to do so, we propose reducing the number of decisions taken by researchers, by automating most of them.

The diagram presented in Figure 2.1 schematically summarizes the proposed methodology. Three main steps can be observed: benchmark instance selection, automatic algorithm generation, and artifact generation. In short, the inputs for the process are the instance data, the implemented algorithmic components (source code), and the optional configuration for the different steps. As for the output, we have all the artifacts that implement the automatically generated algorithms produced during the steps described next, ready to be used by the research community.

In the first step of the methodology, instances are classified according to their structural features, returning a number of representative instances which are selected as benchmark instances. In the second step, benchmark instances are used to combine and configure all available algorithmic components, most of them implemented by the researcher, generating several algorithm proposals. Finally, the last step is responsible for taking the best algorithm configuration, or those selected by the researcher, and generating the final publishable artifacts.

The proposed methodology is in line with the one described in [23]. As principal outcomes, it allows researchers to conduct studies about repeatability, reproducibil-

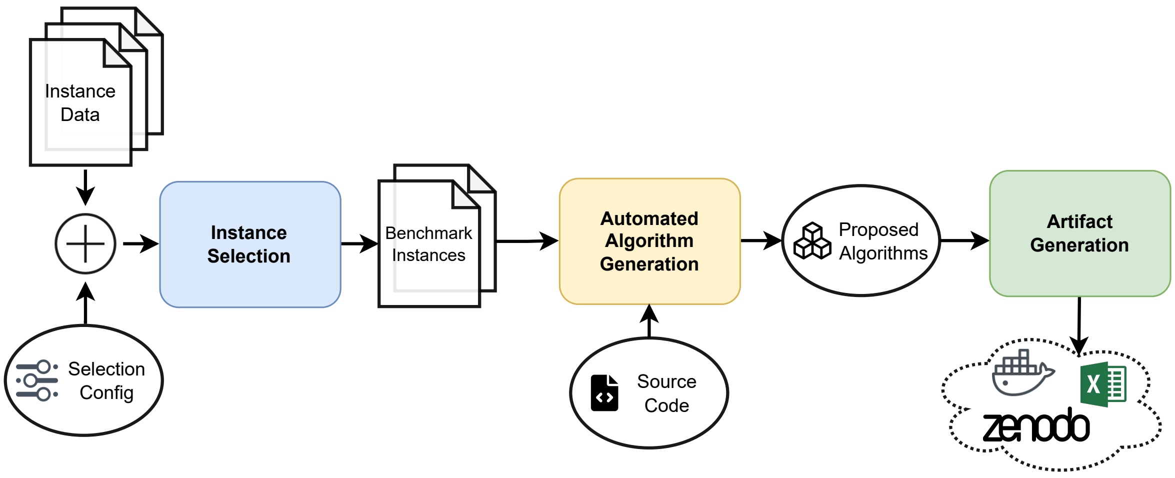


Figure 2.1: Proposed methodology to favor reproducibility.

ity, replicability, and generalizability. Next, the three phases of the methodology are described.

2.2 Instance selection

The most common strategy when dealing with instance data is splitting it into two different subsets, using two different criterion strategies. The first one, commonly used by the ML community, consists on dividing the dataset on a preliminary or "training set", used for training a model, and a "testing set", used for testing its performance with previously unseen data. The second strategy is similar, but, in contrast to the first, the "test set" contains the "training set", or in other words, the full instance data is used when testing the approach performance. As this is the most common strategy used in the heuristic optimization community, we will follow this design in our methodology.

The benchmark instance set size is commonly arbitrarily chosen, but usual criteria include, among others, the total number of instances of the problem, the time required for performing the preliminary experimentation, or characteristics intrinsic to the instances. We will not attempt to determine the correct value for the preliminary set, although a default value of 15% is used in the implementation, leaving it to the researcher to adjust if necessary. Nevertheless, what will be done is automating the whole process given the desired "training set" size.

An automatic benchmark selection pipeline is proposed based on the four aforementioned principles: repeatability, reproducibility, replicability, and generalizability. To this end, the number of decisions based on the researcher's experience is kept to a minimum, by favoring those based on rational criteria. In particular, some techniques commonly used with great success in the ML field are adapted. The idea to apply existing approaches in the ML domain to metaheuristic is an emergent field of research [24–26]. Specifically, we will apply Principal Component Analysis (PCA) and k-means clustering algorithms in the automated instance selection step.

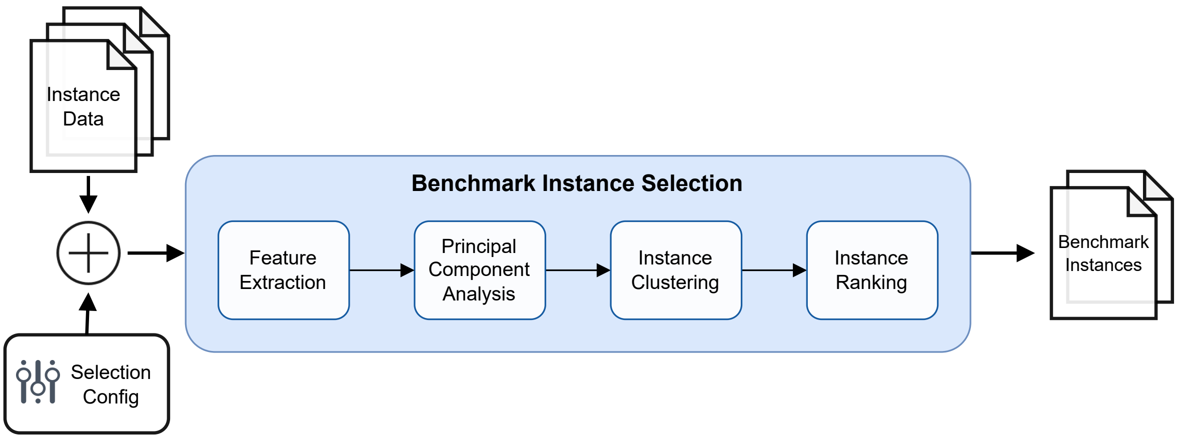


Figure 2.2: Proposed methodology to favor reproducibility, detailed first step.

As seen in Figure 2.2, the benchmark instance selection process receives as input the whole set of available instances of the problem and a user-defined configuration that includes the number of instances to be selected, and, optionally, parameters for the initialization of the PCA and k-means algorithms (seed, confidence level, number of repetitions, etc.). Then, four steps are sequentially executed: Feature extraction, Principal Component Analysis, Instance Clustering, and Instance Ranking.

Feature extraction: instances are parsed and processed to obtain, for each one of them, the set of metrics or features. This step needs the intervention of the researcher, since it requires identifying and describing the features that are relevant for a given problem. Commonly used metrics, such as those applicable to graphs (cardinality, density, etc.) are available, but the researcher must decide which ones to use. In most cases, a deep understanding of the problem domain helps to detect which features can be useful. In case of doubt, it is better to include as many metrics as possible, as redundancies will be removed in the next step.

Principal Component Analysis. In this step, we take the output matrix from the first step, and, after properly scaling the data, to make different properties comparable, we propose to reduce its dimensionality while minimizing the information loss, in order to simplify the subsequent steps. To do so, the usage of two well-known techniques is proposed: standardization of the data and PCA. Thanks to the PCA procedure, the existing properties are replaced by a new set, denoted as the principal components set, that retains most of the information from the original set, with a smaller number of variables [27].

To determine the number of principal components, we use the explained variance ratio, i.e., the percentage of variance that is attributed to each one of the selected components. Generally, we would like to select a number of principal components that explain around the 90% of the data [28]. This parameter can be optionally customized by the researcher if necessarily.

Instance Clustering. In the third step, the k-means algorithm is used to classify the instances in clusters, according to the principal components set generated in the last step. The k-means algorithm is one of the most widely used clustering methods. It aims to partition a set of observations into a predefined number of k clusters, classifying similar elements in the same cluster [29].

A simple and popular solution to automatically determining the recommended number of clusters is the elbow method, which, in short, executes the k-means algorithm for a determined range of k values. Then, for each generated partition in each k executions, the average distortion score (accumulated sum of the squared distances from each point to its assigned cluster centroid) is computed.

Finally, the optimal value of k is determined automatically using the method proposed by [30]. Intuitively, the elbow point is defined as the point where the slope of the curve has a significant change. See Figure 2.3 for an example, where the "elbow" is located at $k = 5$ when plotting the obtained scores, as marked by the dashed line.

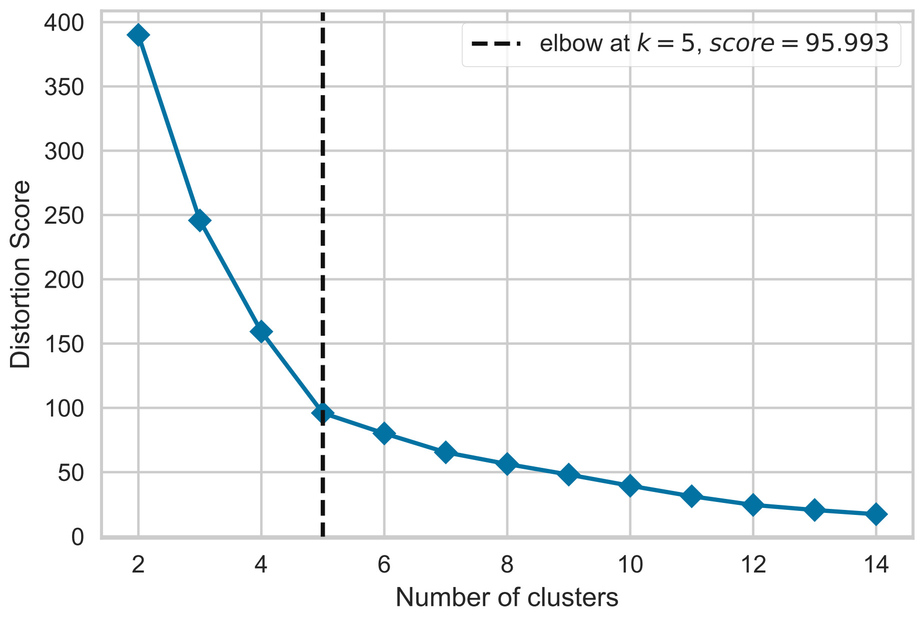


Figure 2.3: Example of distortion scores for $k \in [2, 14]$. The black dash line represents the elbow point.

Instances Ranking. The fourth and final step of the automated instance selection aims to determine what instances will be part of the test or benchmark set. For this purpose, the k clusters generated in the previous step are sorted in descending order according to their size, i.e., the number of instances they contain. Afterward, they are traversed in order, selecting from each of them the instance with the minimum distance to the current cluster centroid. The process is repeated automatically until the benchmark set has the desired number of instances. Thanks to this strategy, it is guaranteed that the preliminary set will contain diverse instances according to their structural features.

2.3 Component driven design

Before explaining the Automated Algorithm Generation step of the methodology, we need to precisely define what is exactly an algorithmic component, and how they may be implemented and described so the next step can properly work with them.

An *algorithmic component* is defined as any procedure or method used in the context of any heuristic or metaheuristic element and any of their required internal components. In other words, any *piece* composing an algorithm could be considered as an algorithmic component. Constructive methods, stopping conditions, neighborhoods, local search methods, and all their respective parameters, are all considered algorithmic components. Note that all heuristics and metaheuristics are considered algorithmic components themselves, as they are always methods that could be used by other metaheuristics, for example by a multistart method.

This component based design is heavily inspired by both the SOLID design principles and the proposals in [31]. In the later, the authors argue that automated design frameworks should be "truly extensible algorithm templates that support reuse without modification". Specifically, algorithmic components are by definition replaceable by any other component that matches the same behavior specification (Liskov substitution principle [32]), for instance, in the case of constructive methods, any constructive implementation should be replaceable by any other constructive; and algorithms may be extended, but they cannot be modified (Open-closed principle [33]). The importance of following this design is highlighted in [34], where it stipulates the requirements for reusing algorithmic components and automated assembly of algorithms.

Any algorithmic component can declare a dependency on any other component type. This dependency will be automatically resolved to any available component which fulfills the required functionality. In this regard, if a component $\mathcal{A}$ requires a component $\mathcal{B}$ in order to work, this dependency can be satisfied either by $\mathcal{B}$ or any other component that matches the same specification as $\mathcal{B}$, by means of inheritance, composition or any other available mechanism of the programming language. In grammar notation, this would be described as in Grammar 2.1. The first rule of the grammar is always the list of available algorithms in the current context, represented by $\mathcal{S}$ with each found algorithm as a derivation. The second rule, specifies that algorithm $\mathcal{A}$ depends on component, represented in the grammar as component $\mathcal{B}$. $\mathcal{B}$ can be subsequently replaced with any component that matches the specification of $\mathcal{B}$, defined as b_1, b_2 … b_n.

In Grammar 2.2, we can see a more complex example, generated from the components in Figure 2.4. In this example, we have two algorithms, $\mathcal{A}$ and $\mathcal{E}$. Both algorithms depend on a component of type $\mathcal{C}$, which may be for example the constructive method.

$$\begin{array}{lcl} \mathcal{S} & ::= & \mathcal{A} \mid \text{Other Algorithms} \\ \mathcal{A} & ::= & \mathcal{B} \\ \mathcal{B} & ::= & b_1 \mid b_2 \mid \ldots \mid b_n \end{array}$$

Grammar 2.1: Example grammar rules generated for replacing $\mathcal{A}$ dependency on component $\mathcal{B}$ with any available implementation that matches the specification.

This constructive behavior is implemented by two components, c_1 and c_2, and any of them could be provided to the algorithms. Moreover, $\mathcal{A}$ requires a component of type $\mathcal{B}$, which can be substituted by b_1, b_2 or b_3; while $\mathcal{E}$ requires a component of type $\mathcal{B}'$, which can only be either b_2 or b_3. One of our main contributions is generating the grammar from the set of automatically detected components and their relationships when the tuning procedure is run, and therefore it does not need to be written by the user.

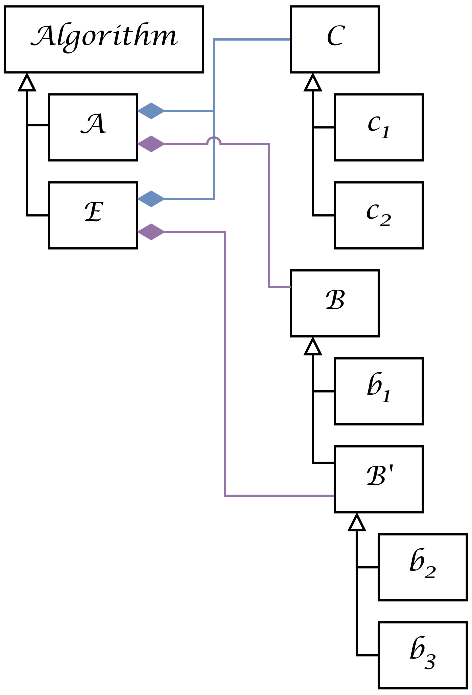


Figure 2.4: Example algorithms $\mathcal{A}$ and $\mathcal{E}$, and their dependency on $\mathcal{B}$ and $\mathcal{B}'$ respectively, and common dependency on $\mathcal{C}$. Empty arrows represent inheritance, while the filled diamond mean that the component at the side of the diamond is composed by or depends on the component at the other side.

Note that recursivity is implicitly allowed, and may occur in cases such as component $\mathcal{A}$ depending on any component of its very same type (direct or tree recursion) or, indirectly if, for example, a component $\mathcal{A}$ depends on type $\mathcal{B}$ which depends on component $\mathcal{C}$, which requires a component of type $\mathcal{A}$, thus creating a loop.

Figure 2.5 shows a subset of the proposed algorithmic components hierarchy based on the usual classification of algorithms, constructive methods, improving methods,

$$\begin{array}{lcl} \mathcal{S} & ::= & \mathcal{A} \mid \mathcal{D} \\ \mathcal{A} & ::= & \mathcal{B}\mathcal{C} \\ \mathcal{D} & ::= & \mathcal{B}'\mathcal{C} \\ \mathcal{B} & ::= & b_1 \mid \mathcal{B}' \\ \mathcal{B}' & ::= & b_2 \mid b_3 \\ \mathcal{C} & ::= & c_1 \mid c_2 \end{array}$$

Grammar 2.2: A more complex grammar generated or algorithms $\mathcal{A}$ and $\mathcal{D}$, and their dependencies. $\mathcal{S}$ represents the start rule.

perturbation methods and neighborhoods. Empty triangle arrows represent the type hierarchy, or in other words, where may a given component may be used. For example, both *MultiStart* and *Simple* are of type *Algorithm*, and may be used anywhere where an algorithm is requested. Furthermore, diamond arrows mean that the component at the side of the diamond is composed by or depends on the component at the other side, that is, a *MultiStart* element is composed by an *Algorithm* component; and *Simple*, *IteratedGreedy*, *VNS* and *SimAnnealing* algorithms need a *Constructive* component. The blue elements represent components developed by the user of this methodology. This components may be added anywhere in the hierarchy, extending it to match the needs of the researcher.

In this figure, components are ordered in three columns to facilitate the explanation. The first column, represents the *Algorithm* component type, which is the only mandatory component that must be used, either by extending it and adding a custom implementation, or by using any of the already implemented algorithms. In the second column, the most common component types used in the literature, namely *Constructive*, *Improver* and *Shake* or perturbation components, are presented. Finally, in the third column, any specific components required depended on by any other component, problem specific components, or additional component implementations from the second column may be found. Users may chose to ignore the existing hierarchy, opting to create their own, and that would be perfectly valid for the methodology.

Note that several examples of recursivity can be seen in Figure 2.5, for instance the *MultiStart* algorithm requires an algorithm over which it will iterate; and the *VND* requires several *Improver* methods, where one could be a *SequentialImpr*, and inside this *SequentialImpr* another *VND* with a different configuration may be used. If recursion is not desirable for a given component, it can be specified in the component metadata, for example in case a *VND* does not want any of its internal *Improver* components to be a *VND*.

In order to provide this metadata, the usage of annotations [1] is proposed. Annotations are part of a program but do not have a direct effect on the behavior of the code where they are used. For example, in a GRASP constructive method, the α value cannot take an arbitrary number, and it is usually limited to the range $[0, 1]$. While it can be automatically detected that the α parameter must be a real number, the

[1]Also known as decorators or attributes, depending on the programming language used.

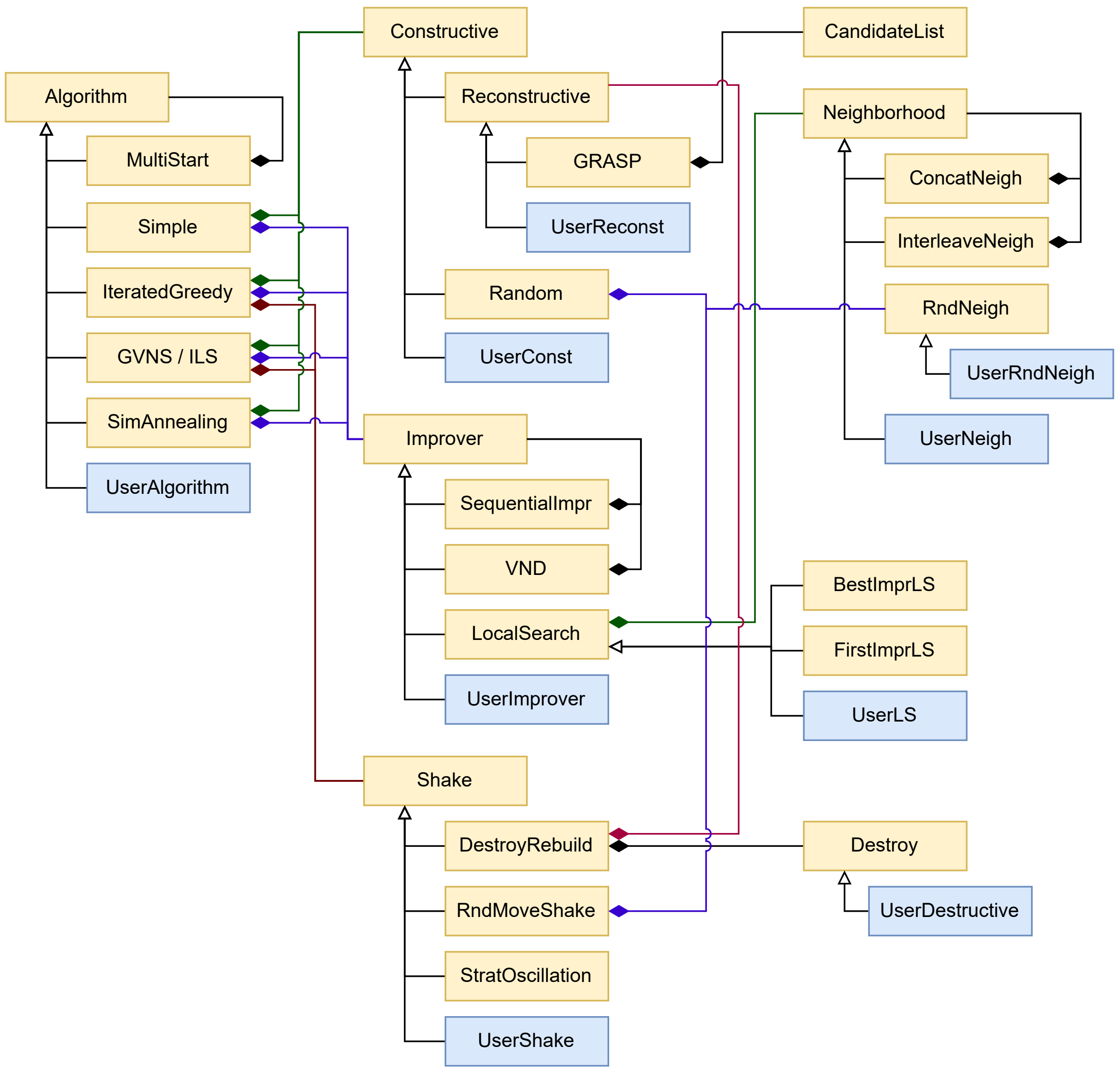
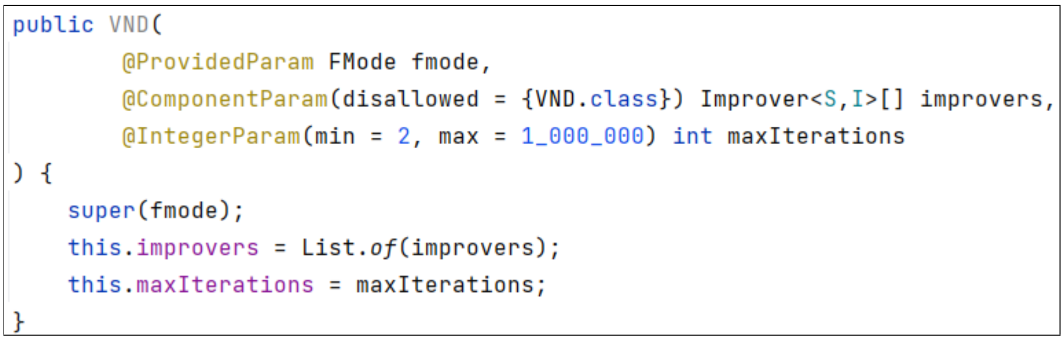


Figure 2.5: Algorithmic component hierarchy provided by the framework. Diamond arrows mean that the component at the side of the diamond is composed by or depends on the component at the other side, while the unfilled triangle represents that a given component implements or is of the type pointed by the triangle. Colors are used to more easily differentiate relationships.

framework cannot infer that its values must be in range $[0, 1]$, or in general, any applicable restriction. Therefore, we propose the following annotations to provide optional metadata for the component dependencies:

- *Algorithmic component parameters*: represents any algorithmic component type. A component may use it to restrict the components that may be used, reducing the number of possibilities available by default.
- *Context parameters*: represents any parameter type whose value is either fixed or calculated at run time by a user provided function. Examples can be the direction of the objective function, or the algorithm name.
- *Integer and real parameters*: represents either an integer or a real value, respectively, allowing the user to define a range of valid values.
- *Categorical and ordinal parameters*: represents a value to be chosen between a predefined set of values. An example of a categorical parameter could be the acceptance criterion used in Simulated Annealing. In contrast to categorical parameters, ordinal parameters imply an order between the predefined values. One example of its application could be if we wanted to restrict the values for the α constructive to any of $\{0, 0.25, 0.5, 0.75\}$, which is a category, but has an explicit ordering.

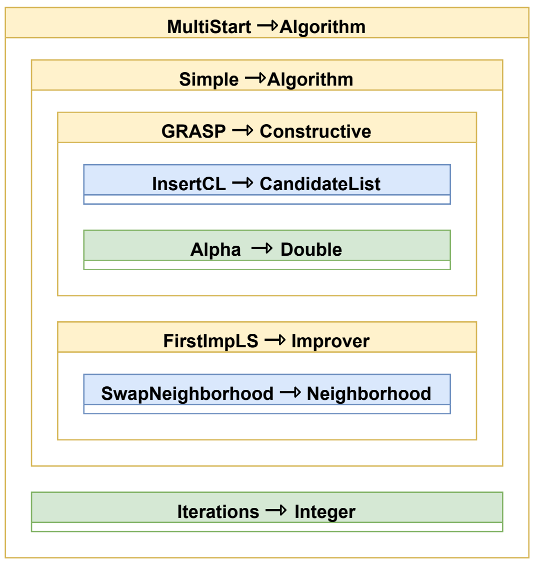


```java
public VND(
        @ProvidedParam FMode fmode,
        @ComponentParam(disallowed = {VND.class}) Improver<S,I>[] improvers,
        @IntegerParam(min = 2, max = 1_000_000) int maxIterations
) {
    super(fmode);
    this.improvers = List.of(improvers);
    this.maxIterations = maxIterations;
}
```

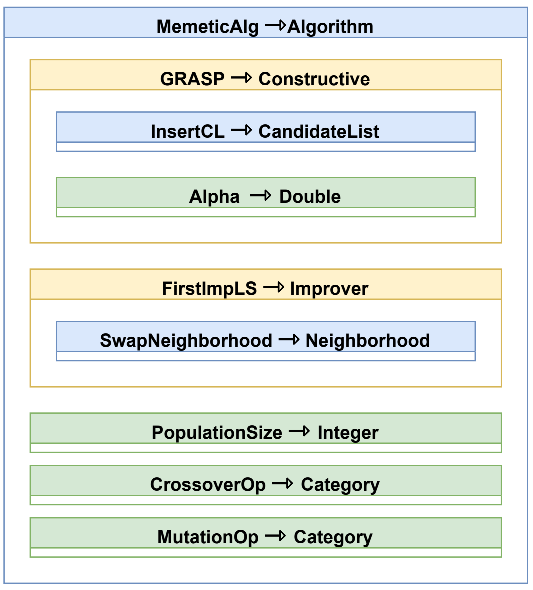
Figure 2.6: Example metadata provided in the constructor method of a VND improver implemented in Java.

In Figure 2.6, an example annotated constructor of a possible VND Java implementation is shown. The `fmode` parameter represents the objective function direction, either maximize or minimize, and therefore is not a parameter that should be considered for the tuning step. The second parameter, `improvers`, receives a sequence of Improver methods, and uses the `ComponentParam` annotation to specify that none of the received improvers should be of type VND, even if VND is a valid improver method. Finally, the `maxIterations` parameter uses the `IntegerParam` annotation to specify that the VND should be always executed a minimum of two times and a maximum of one million.

Finally, two examples of generated algorithms are presented in Figure 2.7, where the notation of yellow and blue colors is also used for framework provided and user



(a) Generated multi-start GRASP algorithm.



(b) Generated hybrid evolutionary approach, using a memetic algorithm defined by the user.

Figure 2.7: Examples of automatically generated algorithms. Each component and parameter is represented as a box, which may recursively contain components. Provided components in the framework are yellow; integer, real, categorical and numerical parameters are green, and blue components are the ones provided by the user.

provided components, respectively. Specifically, in Figure 2.7a, an automatically generated GRASP algorithm can be observed. Note that standard parameters, shown in green background, such as the *GRASP* α value, or the number of iterations for the *MultiStart*, are also considered as dependencies that must be fulfilled. In Figure 2.7b, a more complex example built from a user implemented algorithm is provided. Even though the algorithm is originally unknown to the framework, it will be detected, and its dependencies analyzed at run time. This behavior is the key to the lack of restrictions for the interactions between user components and components provided by the framework, as they are all components, without any differentiation between high level and low level components, or between framework provided and user provided components.

2.4 Automated algorithm generation

Once the preliminary instances are selected, the next step in the methodology is automatically generating algorithmic components configurations, tuning the algorithm components and deciding which combination of them will be used in the final proposal.

Previous works, such as [13], were taken into account in order to design this proposal. There, the authors discuss the problem and propose metaheuristic specific automated design guidelines, noting the lack of a general framework in the literature. Existing structural tuning proposals, such as [35], are usually restricted to applying a limited set of metaheuristic methods in a single problem family. This imposes a rather inflexible algorithmic structure. In order to overcome this inflexibility, recent works, such as Emili [36] propose classifying components into low-level components, which are problem dependent, and high-level reusable components, provided by the framework. Moreover, the usage of a grammar is proposed to model component relationships and the design space [37]. This approach, while functional, requires a lot of user work and is error prone: the grammar must be manually defined, and each available component must be manually integrated in the framework. Another well known framework, ParadisEO, recently added an automated configuration module [38], but, unfortunately, suffers from the same limitations as Emili.

Therefore, one of the main contributions in this step of the methodology, if not the most important, is proposing an strategy for automating the grammar generation and the subsequent design space exploration. In Figure 2.8, the main pieces of the automated algorithm generation are presented. The figure is divided in two main processes: the component discovery and grammar generation (on the left, in yellow background), and the main automatic algorithm design loop (on the right, in orange bakground).

First, both the code written by the user and the framework itself is analyzed, looking for all available algorithmic components, which are cataloged according to the roles they may perform. Due to the fact that this step runs each time a new experiment starts, the user needs neither to enumerate algorithm components nor make any

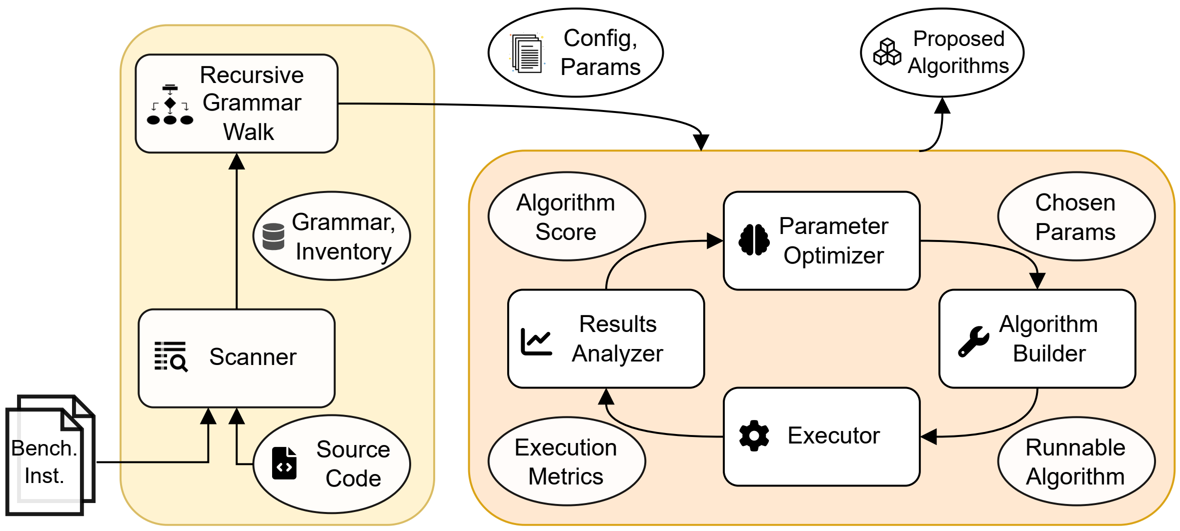


Figure 2.8: Proposed methodology to favor reproducibility, detailed second step.

modification for the new components to be detected. The grammar is generated simultaneously: for each discovered algorithmic component, its dependencies are analyzed and the corresponding grammar rules generated.

Once the grammar has been built, and the inventory of algorithmic components is completed, the next step consists on recursively walking the grammar to explore all available combinations, or in other words, building the derivation tree. As recursivity is allowed, exploration is limited by a maximum depth parameter. If any algorithm component does not generate a valid component at the given maximum depth, the affected derivation tree branches are pruned.

After the derivation tree is complete, a parametric description [39] is built. This description will use categorical parameters to represent a choice among components, and conditional parameters will control which other components (and their parameters) are activated when certain components are selected earlier in the derivation tree. The usage of a parametric description will allow us to use already existing powerful AAC methods, such as `irace` [21], as long as they support parameter spaces with the required characteristics (categorical, numerical and conditional).

Although `irace` is used as the default parameter optimizer in the implementation of the methodology due to its advantages (widely used, best benchmark results [40] in multiple scenarios, open source and well documented), any other parameter optimizer that implements the required functionality can easily replace it, for example, Optuna [41].

The remaining components of the main automated design loop are the *Algorithm Builder*, the *Executor* and the *Results Analyzer*. The *Algorithm Builder* is responsible for transforming the list of parameters and their assigned values, coming from the AAC tool, into an intermediate language representation, which will then be used to instantiate the runnable algorithm. The usage of an intermediate language provides an abstraction layer over the parameter optimizer, making it easy to have multiple parameter optimizer implementations and greatly aiding both debugging and report generation. The runnable algorithm is then submitted to an executor, which will run the algorithm in a reproducible environment with a maximum time limit, reporting the execution metrics, such as how the objective function evolves during the whole run, to the next step. Finally, the results analyzer is responsible for transforming the metrics generated during the algorithm execution into a numeric score that is returned to the parameter optimizer. For a more detailed description, see Chapter 9.

2.5 Artifact generation

According to the definitions from the literature, an artifact is a digital object that was part of the research process, either created or being an external tool used in this process [42]. By this definition, instances, result files and scripts used to analyze results are all considered artifacts.

To implement our proposal, existing technical solutions will be reused. Specifically, tools like Docker, and platforms like the Open Science Foundation or Code Ocean, allow the portability of software to different environments such as virtual machines and containers [43]. By properly using these technical solutions, everything needed to replicate an experiment is included and published. Not only is this useful for validation, as the preservation of code and data may be more useful in the long-term than the availability of a reproducible experimental environment. This is due to the rapid obsolesce of both software and hardware. Even if the original study becomes non-reproducible due to the obsolescence of its original artifacts, studying their code and data could help future replication and generalization efforts.

Therefore, taking into account the previously mentioned four different types of reproducible studies according to [23], our methodology proposes the generation of a number of mandatory artifacts, which are listed below:

- **Problem instances and results files**. Not only the raw instance files to be used in an experiment, but also, explanatory text files regarding technical questions, that may for instance answer questions about the file format used. In addition, plain text files (or spreadsheet files) with the obtained results as used in the publication, along with raw or non-aggregated results should be made available.
- **Source code of the proposed methods**. Additionally to the code of the algorithm components, these files should include any utilities developed, such as those related to instance generation or processing (if needed), or any automated processing of the results.
- **Analysis artifacts**. Generated diagrams, R scripts or any tool used to generate them, with the raw results used to generate any related figure or summary table.
- **Executable artifacts**. Binary or executable versions on a portable environment

for all the elements described in the previous item. The usage of the platforms described at the start of this section is strongly encouraged.

Giving access to these artifacts allows the development of any kind of reproducibility study. As it will be later demonstrated, most of these artifacts are automatically generated by the implementation of the proposed methodology.

Chapter 3

Optimization problems under study

In this chapter, first, a short introduction to metaheuristics methods and common components used to solve optimization problems is provided. Then, in next sections, the specific optimization problems that will be analyzed, solved and compared against the state-of-the-art will be introduced. All chosen problems belong to different problem families, with radically different characteristics, in order to properly validate the methodology and its implementation. For each optimization problem, an introduction is provided, briefly explaining the approached problem, the context in which it appears, followed by its practical applications and mathematical formulation. Finally, the state of the art, taking into account both exact and heuristic methods, is summarized.

3.1 Heuristic and metaheuristic methods

Metaheuristics are one of the most prominent and successful techniques used to solve a large and varied set of complex and computationally hard optimization problems that arise in almost every context, such as economics (e.g., portfolio selection, risk management), industry (e.g., scheduling, logistics) and engineering (e.g., materials optimization, routing). Metaheuristics can be seen as general algorithmic frameworks that with few modifications are able to tackle a variety of optimization problems. Unlike exact algorithms, metaheuristics can neither guarantee the optimality of the obtained solutions, nor, unlike approximation algorithms, define how close the obtained solutions are from the optimal ones. However, they do provide good quality solutions in reasonable computing times for hard and complex optimization problems.

Fred Glover coined the term metaheuristic in 1986 [44], defining it as *"a master process that guides and modifies other subordinate heuristics to explore solutions beyond simple local optimality"*. Metaheuristics constitute a very diverse family of optimization algorithms including methods such as Tabu Search, Greedy Randomized Adaptive Search Procedures, Scatter Search, Variable Neighborhood Search, Iterated Local Search and Multi-start Methods. In addition, bio-inspired metaheuristics began with Genetic Algorithms, and, among others, include different methods such as Simulated Annealing, Genetic Programming, Memetic Algorithms, Ant Colony Optimization, and Grammatical Evolution. See [45] for a detailed survey about metaheuristic methods.

One metaheuristic algorithm that will be implemented for all problems presented in this chapter is Greedy Randomized Adaptive Search Procedure (GRASP). GRASP was originally introduced in the late 1980s [46] but it was not formally defined until 1994 [47]. GRASP uses an iterative strategy, dividing each iteration in two stages: first, generate an initial solution and then, try to improve it. The first phase usually starts from an empty solution, and a Candidate List (CL) of possible elements to be added to the solution is built. There are two commonly implemented variants, *GreedyRandom* and *RandomGreedy*. In the first variant, each candidate is evaluated using a problem dependent function, and those whose value is no further from the best value by a given margin, provided by parameter α, are added to the Restricted Candidate List (RCL). Then, a random element from the RCL is chosen, adding it to the solution and updating the CL. In the second variant, a subset of elements from the CL is chosen as the RCL, and the best element from the RCL is chosen, adding it to the solution and updating the CL. The number of chosen elements in the RCL depends on the α value: choosing all elements would be equivalent to a greedy approach, while picking only one is equivalent to a random method. This process is repeated until the CL is empty. Thanks to the α parameter, the balance between intensification and exploration during the construction phase is controlled, thus increasing the probability of globally finding better solutions. Due to the fact that the solution generated during this constructive phase is unlikely to be a local optimum, a local improvement method is usually applied afterward.

Thanks to GRASP's versatility, there are multiple different algorithms that we can apply in this stage, varying from traditional local search methods to more complex implementations such as hybridization with other metaheuristics that has lead to successful research: Tabu Search [48], Path Relinking [49], Strategic Oscillation [50], or Ejection Chains [51], among others.

3.2 Balanced Minimum Sum-of-Squares Clustering problem

The first problem tackled is part of the clustering problem family. Clustering problems consist in splitting data into set or groups, usually called clusters, so elements in the same group share some specific characteristics or features. In other words, if two elements belong to the same cluster, it is because they are related to each other. Likewise, elements are in different clusters are expected to be barely related. Therefore, in general clustering algorithms try to split the given dataset into several subsets maximizing the similarity of elements in the same subset, while minimizing the similarities between elements in different subsets. Although the similarity metric used is problem specific, one of the most commonly used metric is called sum-of-squares, which gives name to the Minimum Sum-of-Squares Clustering (MSSC) problem family.

Unlike classification problems, where there is previous knowledge of the group to which each element in a set belongs, and therefore data analysis techniques may be applied to learn patterns in a supervised way, in clustering problems no such information is available *a priori* [52]. Designing effective procedures for clustering problems is harder, as the actual group each element actually pertains is not known [53]. Examples of practical clustering applications are pattern recognition, image processing and data mining, among others [54]. Most clustering problems are $\mathcal{NP}$-hard [55], even when considering only two clusters [56].

MSSC considers that the number of clusters k is known *a priori*. The objective of MSSC is to assign a set of n points $P = \{p_1, p_2, \ldots, p_n\}$ located in an s-dimensional Euclidean space $\mathbb{R}^s$ to a cluster $K_i \in \{K_1, K_2, \ldots, K_k\}$. MSSC then tries to find the assignment S of points to clusters with the minimum sum-of-squares distances from each point p_j of the cluster K_i to its corresponding centroid $\bar{p}_i$. Given a cluster K_i, its centroid is defined as the point whose distance to the other points of the cluster is minimum. More formally, the quality of the partition S, denoted as $f(S)$ can be evaluated as:

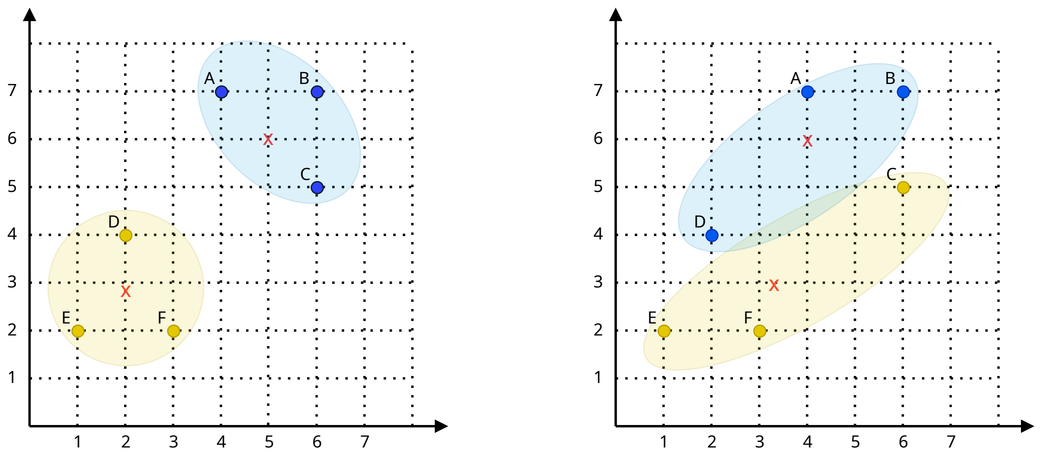
$$f(S) = \sum_{j=1}^{n} \sum_{i=1}^{k} x_{ij} ||p_j - \bar{p}_i||^2$$

where x_{ij} is a binary variable (with $1 \leq i \leq k$ and $1 \leq j \leq n$) that takes on the value 1 if point p_j is assigned to cluster K_i; otherwise, $x_{ij} = 0$. Naturally, this variable satisfies that $\sum_{i=1}^{k} x_{ij} = 1$ for $1 \leq j \leq n$.

In this thesis, we will focus on the Balanced Minimum Sum-of-Squares Clustering (BMSSC) problem, which includes an additional cardinality constraint to the MSSC. Specifically, there will be $n \texttt{ mod } k$ clusters of size $\lceil n/k \rceil$, and $k - (n \texttt{ mod } k)$ clusters of size $\lfloor n/k \rfloor$. BMSSC has also been proven to be $\mathcal{NP}$-hard for $n/k \geq 3$. See [57] for further details.

Figure 3.1 shows an example of two possible clustering solutions for $k = 2$, where we consider the same set of points. On the one hand, Figure 3.1a depicts a solution S_1 conformed with $K_1 = \{\texttt{A}, \texttt{B}, \texttt{C}\}$ and $K_2 = \{\texttt{D}, \texttt{E}, \texttt{F}\}$, resulting in tight clusters. On the other hand, the solution S_2 presented in Figure 3.1b contains clusters $K_1 = \{\texttt{A}, \texttt{B}, \texttt{D}\}$ and $K_2 = \{\texttt{C}, \texttt{E}, \texttt{F}\}$, with a higher objective function, since the sum of the distances between the points and the cluster centroid are higher. In light of this figure, we can conclude that S_1 is a better partitioning than S_2.

As most of the clustering problems have been proven to be $\mathcal{NP}$-hard [58], the majority of the algorithms found in the related literature are approximate procedures. In general, existing approaches can be classified in partitional clustering [59], if the data is divided into disjoint sets and each element is assigned to one set; hierarchical clustering [60], if clusters are created following either a top-down or a bottom-up approach; density-based clustering [61], where elements are grouped following a density function; or grid-based clustering [62], if data is divided into grids with different



(a) Optimal cluster assignment (b) Random cluster assignment

Figure 3.1: Two clustering examples for the same points. Each point has the color of the cluster it is assigned to, whose area is delimited by an oval. The red cross represents the centroid of each cluster.

granularity level. See [63] for a detailed taxonomy of basic and advanced clustering techniques.

First attempts to solve the Balanced Minimum Sum-of-Squares Clustering (BMSSC) problem consider the well-known k-means procedure [64]. This algorithm is a classical clustering method that consists of two main steps: it firstly selects the elements that will become the centroids of the clusters, and then assigns each one of the remaining elements to their nearest centroid. We refer the reader to [65] for an efficient implementation of the method. In orther to minimize the dependency on the initial centroid selection, a new method called k-means++ [66] proposes to mitigate this behavior by choosing diverse centroids. Other k-means variants include the power k-means, which proposes to avoid poor local minima by annealing through a family of smoother surfaces [67]. This algorithm is further improved in [68] when dealing with high dimensions.

Four recent well performing approaches have been identified in the recent literature: a variable neighborhood approach, using the LIMA methodology [69]; a k-means implementation [66] complemented by the Hungarian algorithm [70] in order to satisfy the clusters size constraint; and two adaptations of existing approaches [59, 60] for the MSSC problem to which a repair method is appended in order to guarantee the balance constraint by repairing oversized clusters by reassigning their elements to the best available cluster.

3.3 Vehicle Routing Problem with Occasional Drivers

The Vehicle Routing Problem (VRP) problem family consists on, given a fleet of vehicles, satisfy the demand of a given set of customers, by designing routes limited by the fleet resources, so no customer is left unattended. As can be seen in Figure 3.2, it can be considered a generalization of the TSP, where the depot or origin point can be visited multiple times. In other words, because a vehicle does not need to visit all destinations in a single run, several smaller routes can be created, as long as all customers are covered at least by one route.

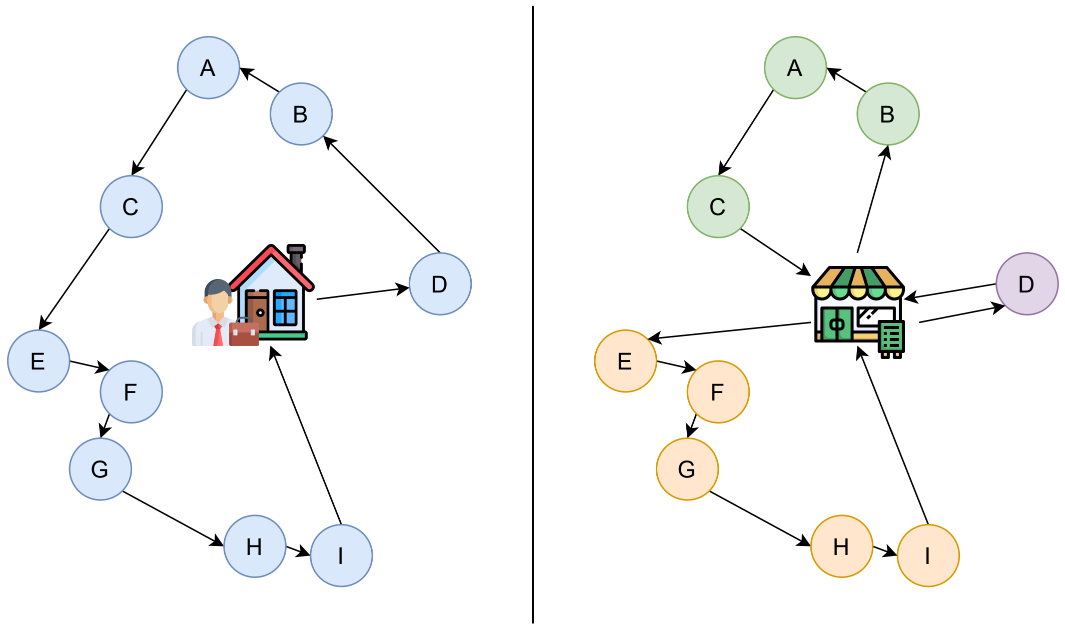


Figure 3.2: Example solutions for the TSP, on the left, and the VRP on the right, given the same set of destinations, represented by colored circles, and origin, represented by the icon. Icons source: Flaticon

Depending on the particular set of restrictions applied, multiple variants of the VRP appear. If, for example, the dimensions or weight of every package are known, and the capacity of each vehicle is considered, the problem is known as Capacitated Vehicle Routing Problem (CVRP). Another variant, the Vehicle Routing Problem with Time Windows (VRPTW), considers that the delivery destinations are only available in a restricted set of hours, for instance during business hours, or when residents are home. See [71] for a taxonomic review of the most common problems in the VRP family and existing metaheuristic approaches.

The explosion in usage of e-commerce platforms in recent years to buy and sell a variety of goods, products and services has lead to an increase in the usage of services provided by logistic companies, specially for the last-mile operations [72]. Last-mile operations, as the name implies, refer to the last step in the logistic process of delivering a product to the consumer, usually from an intermediate warehouse near the final destination. It is estimated that the last-mile cost with respect to the total cost ranges between 13% and 75% [73], and due to this, a heavy focus has been put on trying to minimize its cost. Moreover, the motivation is not limited to a simple cost saving measure. Pressure to increase sustainability has increased in recent years, and as last-mile operations are the least efficient of all logistic steps [74], they are a lot of research is flourishing in the area. We refer the reader to [75] for a review of different urban last-mile distribution strategies in both mature and emerging e-commerce markets.

Two examples of common implemented strategies are using self-service collection and delivery points, also called lockers, where the usage of PIN can provide secure access to goods, and offloading certain costly deliveries to ordinary citizens, also known as crowd shipping, or crowd logistics. The idea behind crowd logistics can be summarized as favoring the participation of ordinary citizens in the logistic process, and while this idea is not new, it has been recently gaining relevance [76–78]. Four different types of crowd logistics are identified by the authors in [77]: crowd storage, crowd local delivery, crowd freight shipping and crowd freight forwarding.

We will only concentrate on crowd local delivery, also known by crowd shipping by other authors [79], which consists on either using other customers to deliver products or hire locals near the delivery route to periodically make deliveries on behalf of the logistics operator. As crowd shipping will minimize the operator logistics network [77], the logistic costs [80], and, eventually, the amount of urban traffic, putting this idea into practice will improve the sustainability of the company. Additionally, the crowd shippers, or occasional couriers, benefit from the approach as it can be an opportunity to earn additional money. Note that even though in the literature crowd shipping dispatchers are usually called vehicles, the transport method is not restricted, examples of it being carpooling, walking, using public transport, etc [81].

The VRPOD can be formally stated as follows. Let $G = (V, A)$ be a complete directed graph, where $V = \{0, K, C\}$ is the set of vertices, with vertex 0 as the depot, $K = \{1, \dots, k\}$ the set of vertices representing the location of the occasional drivers and $C = \{1, \dots, n\}$ the set of vertices corresponding to the location of customers ($|V| = 1 + k + n$). Each node $i \in C$ has an associated positive demand $q_i > 0$. Furthermore, $A = \{(i, j) : i, j \in V, i \neq j\}$ is the arc set, where (i, j) represents a path between vertices i and j. For each pair $(i, j) \in A$, let $d_{ij} \geq 0$ be the length of the shortest path that connects i and j. The cost of a route is the sum of the distances between consecutive nodes, including the depot.

Customers can be served by regular drivers on routes starting and ending at the depot. We consider their vehicles to have a limited capacity Q. This variant of the problem allows hiring occasional drivers to make a single delivery to a customer if the following condition is satisfied. An occasional driver $k \in K$ can serve customer $i \in C$ if $d_{0i} + d_{ik} \leq \zeta d_{0k}$ with $\zeta \geq 1$. In other words, if the extra distance to get the occasional driver from the depot through the customer i is less than or equal to $(\zeta - 1)$ times the direct distance from the depot to the occasional destination's location; $d_{0i} + d_{ik} - d_{0k} \leq (\zeta - 1)d_{0k}$. Therefore, ζ is referred as the flexibility of the occasional drivers. For example, in Figure 3.2, the route that serves customer D may be replaced by any occasional driver, since it serves only one customer, as long as there is at least one occasional driver whose destination is near customer D. It is important to emphasize that a trip of an occasional driver is measured as the distance traveled from the depot to the customer and from the customer to the occasional driver location. Furthermore, it is assumed that the capacity of any occasional driver is enough to satisfy the demand of any single customer, but each occasional driver may only serve a maximum of one customer.

The objective of the VRPOD is to minimize the aggregated cost incurred by using a combination of both regular and occasional drivers. Note that an occasional driver is paid only if they serve a customer. The occasional driver payment can be calculated taking into account the compensation rate, denoted by ρ, and the the compensation scheme used. Two compensation schemes are presented in [82]. In the first one, the compensation does not depend on the occasional drivers' destination. Thus, every occasional driver receives ρd_{0i} as compensation for making a delivery to a customer i. In this scheme, the compensation rate is limited to $0 < \rho < 1$. Therefore, this scheme only requires knowing the location of the customers, which means that occasional drivers serving customers far from their locations are not compensated for the extra mileage incurred. In the second compensation scheme, each occasional driver k receives a compensation of $\rho(d_{0i} + d_{ik} - d_{0k})$ for the extra mileage incurred for serving the customer i, with $\rho \geq 1$. This variant is more difficult to put into practice since the company needs to know the destination of the occasional drivers. For further details, see [82] and [83].

Previous approaches to this problem include both an exact model and an hybrid matheuristic [82]. The exact method, implemented using CPlex, is able to solve to optimality instances with sizes up to 25 in under an hour. The matheuristic method proposes assigning occasional drivers to customers using a relaxed integer programming model, while building traditional routes using a combination of variable neighborhood search and tabu search. This is repeated in a multi-start fashion up to a configured number of maximum iterations. Using the matheuristic approach, the authors are able to solve instances up to size 100 in a few seconds.

3.4 Double-Row Facility Layout problem

The Facility Layout Problem (FLP) family encompass a great variety of optimization problems whose main objective is determining how to distribute a predefined set of facilities given a layout restriction. This problem family is of special interest in the context of Flexible Manufacturing Systems (FMS), where an automaton transports material or performs certain manufacturing operation between different facilities positions [84]. Among the layouts found in the related literature, the SRFLP stands out as one of the simplest layouts, due to the fact that facilities are located all next to each other forming a line [85, 86]; and the Multi-Row Facility Layout Problem (MRFLP), where facilities can be organized in multiple lanes [87]. Other variants, in no particular order, are the circular layout [88] and the T-row layout [89]. For an in-depth survey of facility layouts, see [90].

In this problem family, we will focus on the Double Row Facility Layout problem (DRFLP), which is a variant of the MRFLP where only two rows can be used. The DRFLP is an $\mathcal{NP}$-hard optimization problem. A variant of this problem, that will be studied in Section 4.4, is the Space-Free Double Row Facility Layout Problem (SF-

DRFLP), where, as the name indicates, no empty spaces are allowed between facilities.

The problem can be formally defined as follows: let F be the set of facilities (with $n = |F|$), where each facility $i \in F$ has an associated length l_i, and w_{ij} is the amount of flow between two facilities $i, j \in F$. Then, an instance I of the DRFLP is defined by a triplet $I = (F, L, W)$, where L is a vector (of size n) containing all the facility lengths and W is a squared matrix (of size $n \times n$) of pairwise flows. All the facilities in F must be placed at some position x_i but with no overlapping between consecutive facilities in each row of the layout.

A solution to this optimization problem is to find an optimal mapping π that assigns the set of facilities F to the corresponding layout with two rows, together with the exact horizontal location of the center of each facility i in the layout (abscissa x_i). Specifically, given a facility $i \in F$, $\pi^k(p) = i$ indicates that i is located at position p of the permutation corresponding to row k in the layout π. Hence, the complete layout is defined by two permutations of facilities, one for each row, $\pi = \{\pi^1, \pi^2\}$. It trivially holds that the number of facilities of a feasible solution is $|\pi| = |\pi^1| + |\pi^2| = n$. In addition to the permutation, the location of centers x_i must be provided since free space can be found in a solution.

Then, given an instance I and a solution defined by a permutation π together with the centers x_i of each facility $i \in F$, the objective function of the DRFLP, denoted as $\mathcal{F}(I, \pi)$, is computed as the total weighted sum of the center-to-center distances between each pair of facilities. This cost, usually known as the material handling cost, can be formulated in mathematical terms as it is shown in Equation (3.1).

$$\mathcal{F}(I, \pi) = \sum_{1 \leq i < j \leq n} w_{ij} \cdot |x_i - x_j| \tag{3.1}$$

The optimization problem then consists in finding the permutation π together with the centers x_i of each facility $i \in F$ that minimizes the aforementioned objective function subjected to the following constraints to avoid the overlapping between consecutive facilities in each row:

$$x_{\pi^k(p)} \geq x_{\pi^k(p-1)} + \frac{l_{\pi^k(p-1)}}{2} + \frac{l_{\pi^k(p)}}{2} \tag{3.2}$$

In Figure 3.3, an example solution for the DRFLP is presented. If the length of each facility guaranteed to be an integer greater or equal to 1, it can be demonstrated that the allowed free spaces between consecutive facilities in the optimal solution (if they exist) is a multiple of 0.5 units. This is due to the fact that facilities center position increase by 0.5 for each unit of length, making 0.5 the smallest gap possible between two facilities. Hence, the solution may be represented using a permutation of facilities, as long as enough dummy facilities are present, removing the centers x_i as decision variables, and therefore vastly simplifying the optimization problem.

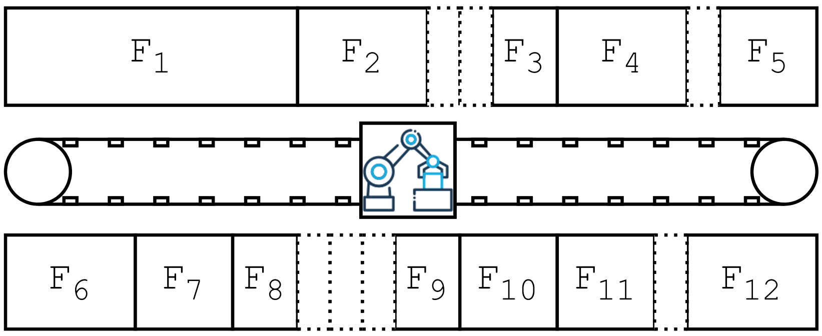


Figure 3.3: Example solution representation including “dummy facilities”, with dotted lines.

The DRFLP was firstly tackled from a heuristic point of view in [91]. In this work, five heuristics were compared with a branch & cut algorithm solving a MIP model. One of the heuristics was finally used to generate an initial solution for the exact algorithm, and its behavior was compared in a set of 16 instances from 6 to 36 facilities. The DRFLP was later studied in [92], proposing a new mathematical model simpler and more performant than the existing one. Some years later, the mathematical model was further enhanced [93], improving the results in instances up to 16 facilities. In [94], the authors propose another mathematical model obtaining competitive results for instances up to 16 facilities. Following his work on this problem, Amaral recently published a work improving the mathematical model once again [95] reporting better results than [94] and, finally, a different work proposing a two-phase algorithm which merges a heuristic method with a linear programming routine [1]. This is the paper with the highest number of instances and, in addition, with the largest number of facilities in the state of the art.

While multiple previous works have proposed both exact methods and heuristic approximations for this problem, reproducibility has been largely ignored at multiple levels. The small size of most instances makes tuning algorithms hard, as most approaches will reach the optimal value relatively easy. Moreover, no effort has been made to group and publish all instances, and no implementation has been published for the proposed heuristic methods. for this problem, and some of them have included heuristic methods.

Chapter 4

Joint discussion of results

In this chapter, the usage of the methodological framework proposed in Chapter 2 is demonstrated against the optimization problems presented in Chapter 3, which belong to completely unrelated families. First, for each problem, the concrete metaheuristic approaches and components implemented are listed, as introduced in 2.3. Then, the results obtained are compared against the state of the art. Lastly, the best previous built approach is compared against the automatically generated proposal. Following the guidelines in Section 2.1, all artifacts are made freely available, and their complete list is provided.

4.1 Results for the Balanced Minimum Sum-of-Squares Clustering problem

For the BMSSC, we propose a Greedy Randomized Adaptive Search Procedure (GRASP) [47] combined with Strategic Oscillation (SO) [96]. Figure 4.1 shows the complete scheme of the proposed algorithm, where we can identify two main phases: GRASP and SO. The first one performs a predefined number of constructions followed by a local search method. Then, the best solution found is further improved in the SO block by alternatively considering feasible and unfeasible solutions. The SO block finally returns the best solution found during the search.

For this problem, three neighborhoods are implemented: adding a point to a cluster, swapping two points of different clusters, and moving a point from one cluster to another. The first move is used during the constructive phase by the GRASP proposal, where points are sequentially added by means of the candidate list until the solution becomes feasible. One remarkable thing is that instead of performing a random cluster initialization, as it is commonly done in the existing literature, we have implemented a new initialization criterion which assigns only one random point to the first cluster, and then computes the minimum distance for every unassigned points to all clusters, choosing the maximum value. This is repeated until all clusters contain exactly one point. This partially built solution is then fed to the GRASP constructive

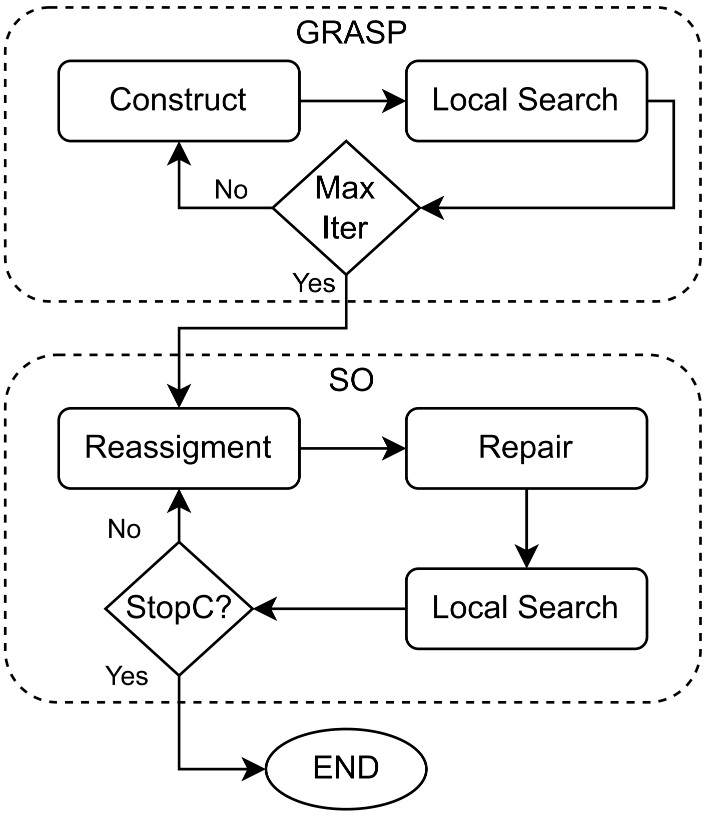


Figure 4.1: Complete scheme of the proposed algorithm.

method.

The second move is used by the local search component, which allows us to improve the solution without breaking the problem constraints. Two different variants are tested, a first improvement approach, where the first swap move that improves the score is applied, and a best improvement approach, where all possible swaps are tested first, and the most favorable one is executed.

Finally, the third movement reassigns one point from its cluster to a different one. According to the problem definition, all clusters have a fixed size, so this move could only be applied in some edge cases by default, such as when the number of points is not divisible by the number of clusters and therefore a limited number of clusters are allowed to have a single extra point assigned. Due to problem limitation, we propose to expand the search space explored, by considering unfeasible solutions, which may eventually lead us to better feasible solutions, which is the main idea behind the Strategic Oscillation (SO) methodology, originally proposed in [96]. Specifically, we will relax the cluster size constraint, by a percentage defined by the β parameter, with range $[0, 1]$. After the move neighborhood is exhausted, i.e, there are no valid moves that improve the current solution, a repair method is required to make the solution feasible again, by reducing the number of elements in all oversized clusters. To do so, we calculate the cost of moving each point from any of the oversized clusters to an undersized one, and execute the move with the minimum cost, until all clusters are correctly sized. As the repaired method returns solutions that are feasible, but may not be local optima, the local search procedure is executed immediately afterward. For a completely detailed algorithm description, see Chapter 6.

In order to perform a fair comparison, the same 16 instances previously used in the literature are considered, which are publicly available in the machine learning repository of the University of California[1]. Additionally, as the instance set is relatively small, 9 new instances from the same repository are considered, for a total of 25 instances. Instance data comes from real world scenarios, such as breast cancer patients, phone sensors, water treatment and accelerometers, plus the metadata required for this problem, such as the cluster size. See [69] for a more detailed description and deep analysis of the instance data.

In this work, we focused on statistically analyzing the performance and stability of the proposal and its parameters. The algorithm is manually built after an in-depth preliminary experimentation, deciding in each step a subset of the required parameters, requiring a lot of manual work. Once all parameters have been fixed, we execute the algorithm 30 times using different seeds, and plot the objective function variation. We depict in Figure 4.2 the associated box and whisker plot, reporting for each instance quartiles, median, minimum, and maximum values.

Our proposal, designed from scratch, is compared against four previous algorithms from the state of the art, named HCOT [60], CGRASP [59], KMH [97] and VNS-LIMA [69]. Table 4.1 summarizes the results for all instances using the Friedman

[1]`http://www.ics.uci.edu/~mlearn/MLRepository.html`

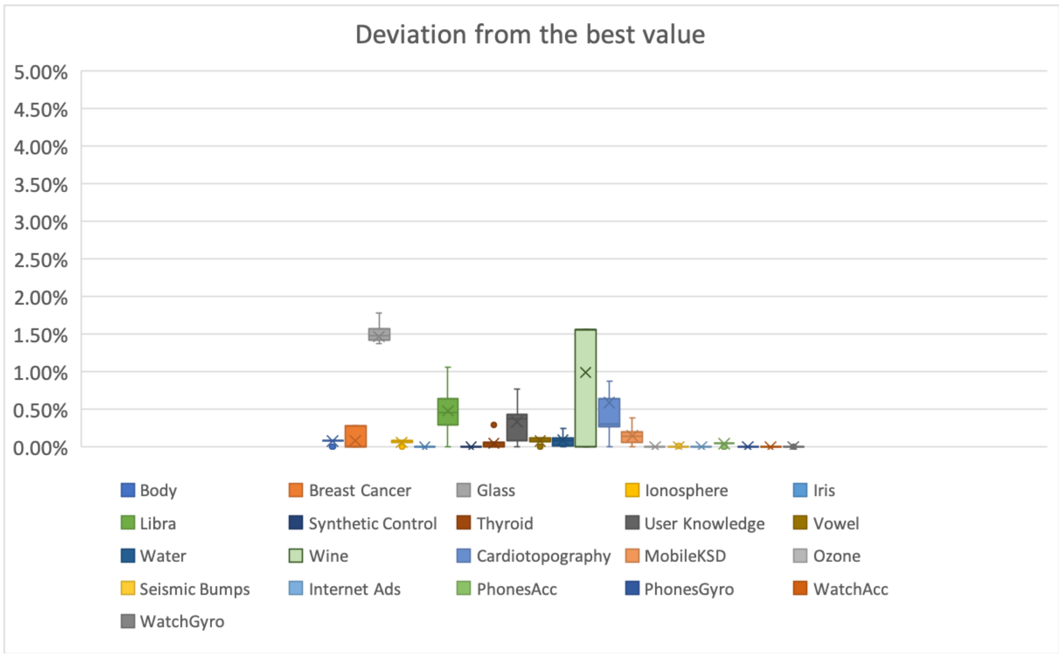


Figure 4.2: Box and whiskers plot using the sum-of-squares metric of 30 independent executions of the proposal.

test, applying the same run time limit value for all the proposals. Additionally to the target objective function for the problem, namely Mean Squared Error (MSE), another commonly used metric not using during the preliminary experimentation, the well-known Davies-Bouldin index (DBI) [98], is reported. The SO proposal obtains the best ranks in both metrics, and according to the p-value, smaller than 0.0001, the results are statistically significant, which confirms the superiority of the proposal.

Metric	HCOT	CGRASP	KMH	VNS	OS	$p < 0.01$
MSE	4.38	3.62	3.50	2.10	**1.40**	YES
DBI	4.34	3.62	3.30	2.10	**1.64**	YES

Table 4.1: Friedman test for both MSE (Mean Squared Error) and DB (Davies–Bouldin) metrics.

Moreover, a sensitivity analysis is conducted. For each parameter of the SO algorithm, namely α (the balance between greediness/randomness of the constructive method) and β (the percentage of cluster size increment inside Strategic Oscillation), different values combinations are tested, fixing the remaining parameters to the best values found in the preliminary experimentation. In order to ensure the analysis is robust, 30 independent iterations of each algorithm were executed. In Table 4.2 the resulting values are presented. The Wilcoxon signed rank test is used to determine the existence of significant statistical differences between variants, in terms of the MSE metric. p values are reported, and are interpreted as follows: a high p value means that there are not statistical differences, while values smaller than 0.05 mean that there is a statistically significant difference. Therefore, according to these results, while the rest of parameters are fixed, we can conclude that changing the α parameter does not have a significant impact on the results, while varying the β parameter does.

α	0.25	0.50	0.75
0.50	0.686		
0.75	0.181	0.581	
RND	0.196	0.123	0.742

β	0.10	0.25	0.50
0.25	0.012		
0.50	0.002	0.000	
0.75	0.133	0.019	0.002

Table 4.2: Sensitivity analysis for numeric parameters α and β.

Finally, to further investigate the performance of the proposed SO procedure, a convergence analysis using Time-to-target Plot (TTTPlot) is performed, proposed by [99]. By using 30 independent runs changing the seed for each iteration, and recording how the objective function evolves in each iteration, we can plot a graph where each value in the abscissa axis represents a running time, while each value in the ordinate axis, reports the probability of obtaining the best-known value. Additionally, the theoretical exponential distribution is also plotted as a continuous line. In Figure 4.3, we can observe that the SO follows the expected exponential run-time distribution for all tested instances. Moreover, in only a few seconds, we have a very high probability of finding the best known value.

4.2 Results for the Vehicle Routing Problem with Occasional Drivers

For the VRPOD, introduced in Section 3.3, an Iterated Local Search (ILS) algorithm is proposed [100]. The ILS proposes coupling a local search method with a shake or perturbation procedure, in order to allow the local search to escape from local optima. This approach is easily parallelizable, and we propose the application of a cooperative scheme, using a multi-start approach. This idea is not new, and has been previously used with great success, for example in [101].

Figure 4.4 summarizes the cooperative scheme. The workload is divided into n workers, named ILS_1 to ILS_N. Each worker uses different parameter configurations for each component, and may even use different component types. Specifically, each worker builds an initial solution, and executes the ILS loop, which consists of iteratively applying a perturbation method followed by a local search method, until a termination criterion is met. Afterward, each worker pushes its working solution to an outgoing FIFO queue and polls the next solution, forming a ring topology. We define *round* as the journey taken by a solution visiting each worker exactly once, and use it as the global stopping criterion. For example, a value of 3 rounds means that each solution will visit each worker three times.

In order to generate both varied solutions and good-quality ones, the GRASP methodology, already introduced in 3.1, is used. GRASP has two important advantages

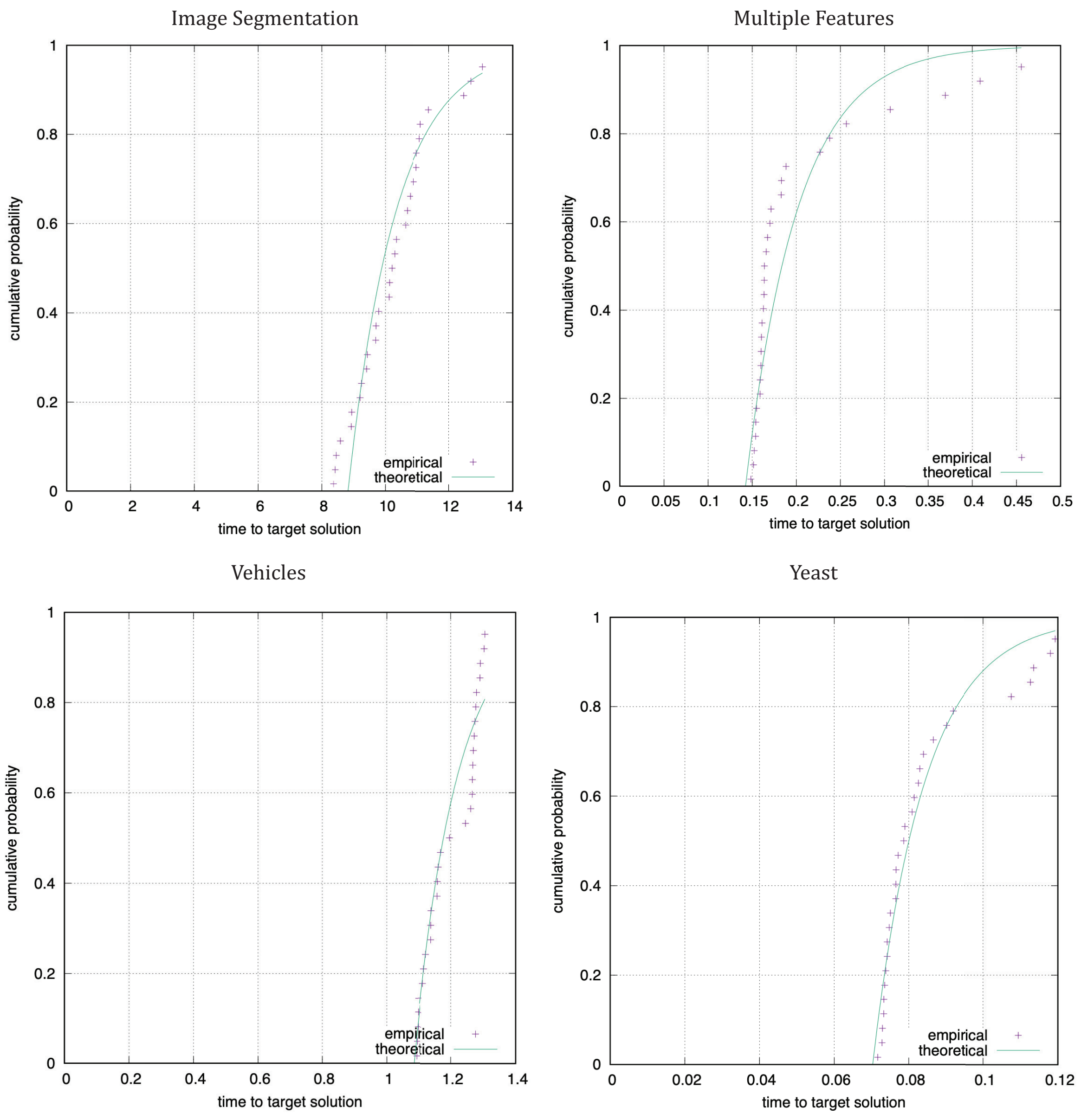


Figure 4.3: TTTPlot of the SO proposal using preliminary experiment instances.

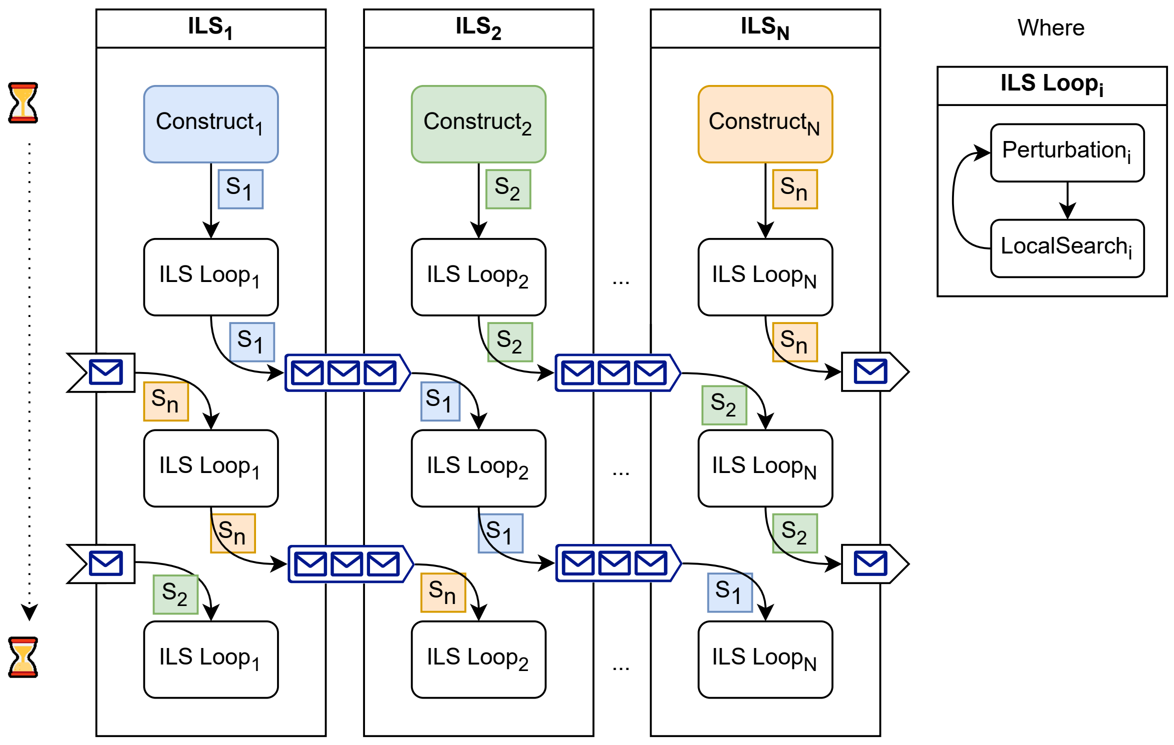
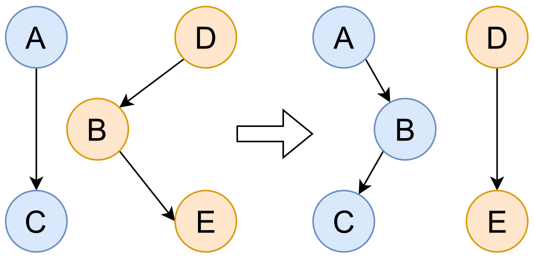


Figure 4.4: Proposed parallel cooperative scheme for the ILS.

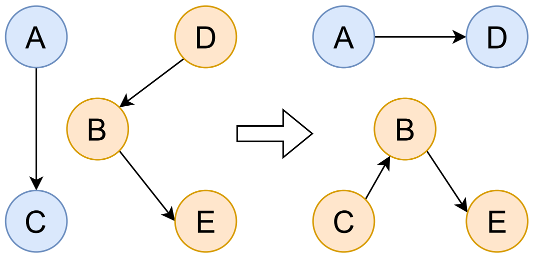
when used with the ILS proposal: by tuning the α parameter, we are able to balance the greediness and randomness of the initial solutions, and its simple design makes possible to obtain large number of feasible solutions using short computing times. Due to the fact that the ILS procedure executes a local search procedure during its main loop, after the perturbation, only the randomized construction phase of the GRASP methodology is used.

We have implemented five different neighborhoods that may be used during the local search step, represented in Figure 4.5. Three neighborhoods are classical in route problems: moving one client from one route to another, or changing its position in the same route (Figure 4.5a); swapping the position of two elements, either in the same or different routes (Figure 4.5b) and reversing a route fragment, usually known as *2-Opt* (Figure 4.5c). Additionally, two neighborhoods are specific for the occasional drivers variant [82]: removing a client from a route and assigning it to an occasional driver (Figure 4.5d) and the reverse operation, removing and occasional driver and assigning it to a route in any position (Figure 4.5e). The neighborhoods may be combined sequentially, or by joining them to form unique neighborhoods combinations. Moreover, the local search can run using different strategies, such as first improvement, or best improvement. Multiple alternatives are considered in the preliminary experimentation, see Chapter 7 for a full detail of all preliminary experiments conducted.

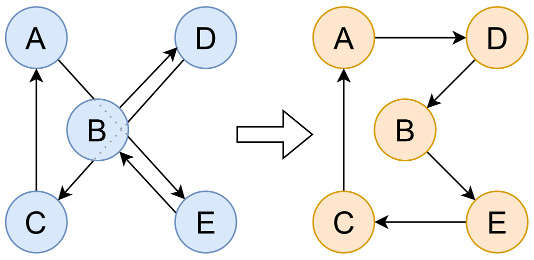
The last component type implemented for this problem is the perturbation procedure. All three perturbation methods have an input parameter which controls the strength of the perturbation, named β. Due to the fact that depending on the per-



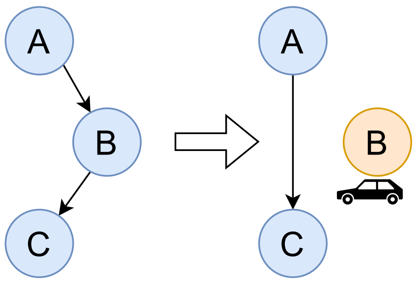
(a) Reassign one client from one route to another. Includes changing position in same route.



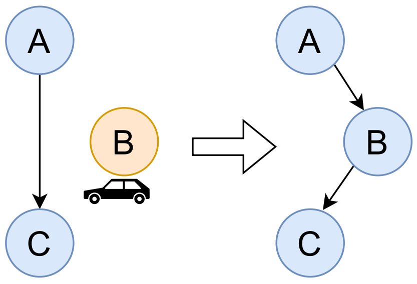
(b) Swap two clients, both between routes and in same route.



(c) *2Opt*: Reverse a route fragment. In this example, the fragment E-B-D becomes D-B-E.



(d) Remove a client from a route and assign it to an occasional driver.



(e) Remove an occasional driver and assign to route.

Figure 4.5: Implemented neighborhoods for the VRPOD.

turbation method, the parameter has a different meaning, we have to carefully select its values during the tuning step. We have considered three different strategies, taking into account all the components already implemented:

- ***RandomMove*:** Selects and executes β random moves from any of the five previously defined neighborhoods, ignoring the performance impact over the objective function.
- ***RouteCost*:** Sorts all routes by their cost per customer, and destroys β random routes, removing all its customers, using a weighted random distribution, i.e, the more costly a route per customer is, the more likely it will be destroyed. This destructive is inspired by existing approaches, such as [102] and [103]. Afterward, the solution is repaired using a reconstructive method, in order to reassign all removed customers and make the solution feasible.
- ***RandomDeassign*:** Randomly selects and removes β customers according to a uniform distribution. Afterward, the removed customers are reassigned using a reconstructive component, to make the solution feasible again.

For both the preliminary and the final comparison against the state of the art, the complete instance set from [82] is used. All components configurations are chosen according to a sequential experiment design. For a full detailed description of the preliminary experiments, see Chapter 7. The best results obtained during the sequential experimentation are obtained by using 100 iterations of ILS, using the *RandomMove* perturbation method, with $\beta = 50$, a GRASP constructive method biased towards greediness, and a best improvement local search method using an extended neighborhood formed by the five proposed neighborhoods.

Table 4.3 compares the results obtained by the proposed ILS algorithm, using the traditional approach (ILS), a simple parallelization approach (ILS_P) and the cooperative scheme (ILS_M). The state-of-the-art results are represented as $SOTA$. Results are summarized by instance size using parameter K, which represents the total number of occasional drivers available. For each algorithm, the following metrics are reported: averaged cost, sum of best values (#B.) and average execution time in seconds (T(s)). The execution time is not reported for the $SOTA$ algorithm, as it has not been made available. Best result per property is highlighted in bold. As expected, the simple parallelized version is the fastest for all instances. For sizes 13 and 25, the three ILS versions find all best known values. For sizes 50 and 100, the cooperative scheme obtains the best results in reasonable computing times. The state-of-the-art falls behind with a total of 158 best values, against the 267 obtained by the cooperative proposal.

In this problem, the key parts of the implemented methodology are the reproducible parallelization and the component driven design. Thanks to the design of a customized and fast performance random manager, we can guarantee exact results reproducibility, in this and all future tackled problems. No matter which thread executes which part of the algorithm, consistence is guaranteed as long as the proposed *RandomManager* API in the framework is used. Moreover, a reusable set of components

	ILS			*ILS_P*			*ILS_M*			SOTA	
$\|K\|$	Cost	#B.	T(s)	Cost	#B.	T(s)	Cost	#B.	T(s)	Cost	#B.
13	**231.46**	**30**	8.03	**231.46**	**30**	**2.69**	**231.46**	**30**	21.26	232.01	16
25	**222.70**	**30**	13.04	**222.70**	**30**	**3.51**	**222.70**	**30**	5.29	222.83	27
50	549.75	52	451.99	549.54	64	**138.81**	**548.54**	**74**	248.95	552.63	50
100	451.48	95	540.63	451.50	97	**106.59**	**449.67**	**133**	294.29	451.19	65

Table 4.3: Summary performance comparison for the proposed *ILS* approach and its variants against the state of the art for the VRPOD.

developed, using the specification presented in Section 2.3.

4.3 Results for the Double-Row Facility Layout Problem

As seen in Section 3.4, previous works in the literature have applied heuristic methods [1] with great success. Our proposal goes one step further, by applying the Iterated Greedy (IG) metaheuristic [104], which has been successfully applied to a great variety of optimization problems [105–108], by combining it with the GRASP methodology, previously introduced in Section 3.1.

Figure 4.6 summarizes the proposal. The IG metaheuristic is represented by the blue box on the right, executing a constructive method first, and a loop of destruction, reconstruction and local search afterward. Note the existence of the *Dummy Manager* component on the left, which will be responsible for adding the necessary dummy facilities to the instance data before the algorithm runs. Initially, the instance will be solved as is, but for each fully completed iteration of the IG metaheuristic, the number of dummy facilities inserted into the facility data is incremented, until the configured termination criteria is met. This way, multiple strategies for using the dummy facilities, such as linearly incrementing them in each iteration, or using an exponential function such as Fibonacci, can be easily tested.

Three different constructive methods are proposed for the DRFLP. The first one generates a random permutation of the facilities, and its main utility is serving as a baseline for comparing more complex approaches, as it is extremely fast. The second and third one are based on the GRASP constructive methodology, introduced in Section 3.1, using different strategies for the CL. Specifically, the second one will consider adding each facility in any position in any row, sorting elements by the cost increase of the insert operation. On the other hand, the third constructive method will consider facilities as pieces to be merged, and in each step will try to merge two facilities groups. Therefore, the CL is formed by all the merge possibilities of the current available pieces or groups, and sorted according to the cost increase of merging them. Figure 4.7 shows all implemented ways to merge two facilities groups. Both the second and third constructive can either run in a *GreedyRandom* or *RandomGreedy* configuration, depending on the parameter configuration.

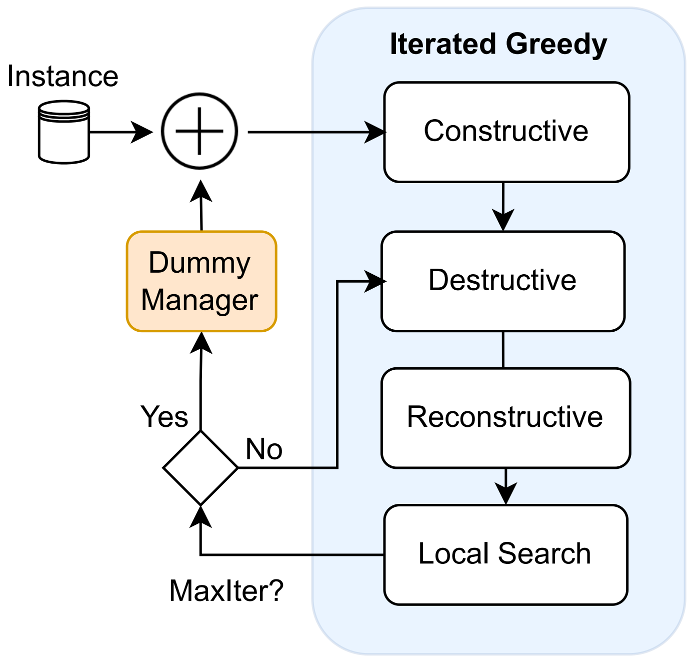


Figure 4.6: Proposed algorithm for the DRFLP.

As shown in Figure 4.6, the IG proposal includes a local search procedure to improve the rebuilt solution. Specifically, we have implemented the neighborhoods for the local search depicted in Figure 4.8, namely moving a facility from one position to any other (4.8a), swapping the position of two facilities (4.8b), either of the same or different rows, and reversing a fragment of a row, which may resemble the 2*Opt* operator usually found in routing related problems (4.8c). Additionally, a variant of the swap neighborhood, using the efficient formulation available in [87] is provided. The local search method may use any combination of neighborhoods using either a first improvement or best improvement strategy.

Moreover, we have implemented two destructive methods. The first one, *D1* consists in randomly removing a number of facilities, determined by the β parameter, while the second one, *D2*, tries to split the layout into several pieces. In order to do so, first, a random facility is selected from any of the rows, and then, any of the vertically overlapping facilities with the first selected one is randomly chosen. Each operation splits the solution into three pieces. For example, in Figure 4.9, where F_2 is first randomly chosen, and then the overlapping F_9 in the other row is selected. After executing the split, the group is split in three: facilities located on the left of the selection (4.9.a); both selected facilities (4.9.b); and facilities located at the right of the selection (4.9.c).

As for the reconstructive methods, we can reuse any configuration of the already proposed constructive methods. Specifically, the second constructor based on facility insertions works best when paired with the *D1* destructive method, while the third constructor based on facility groups is a great match for the *D2* destructive.

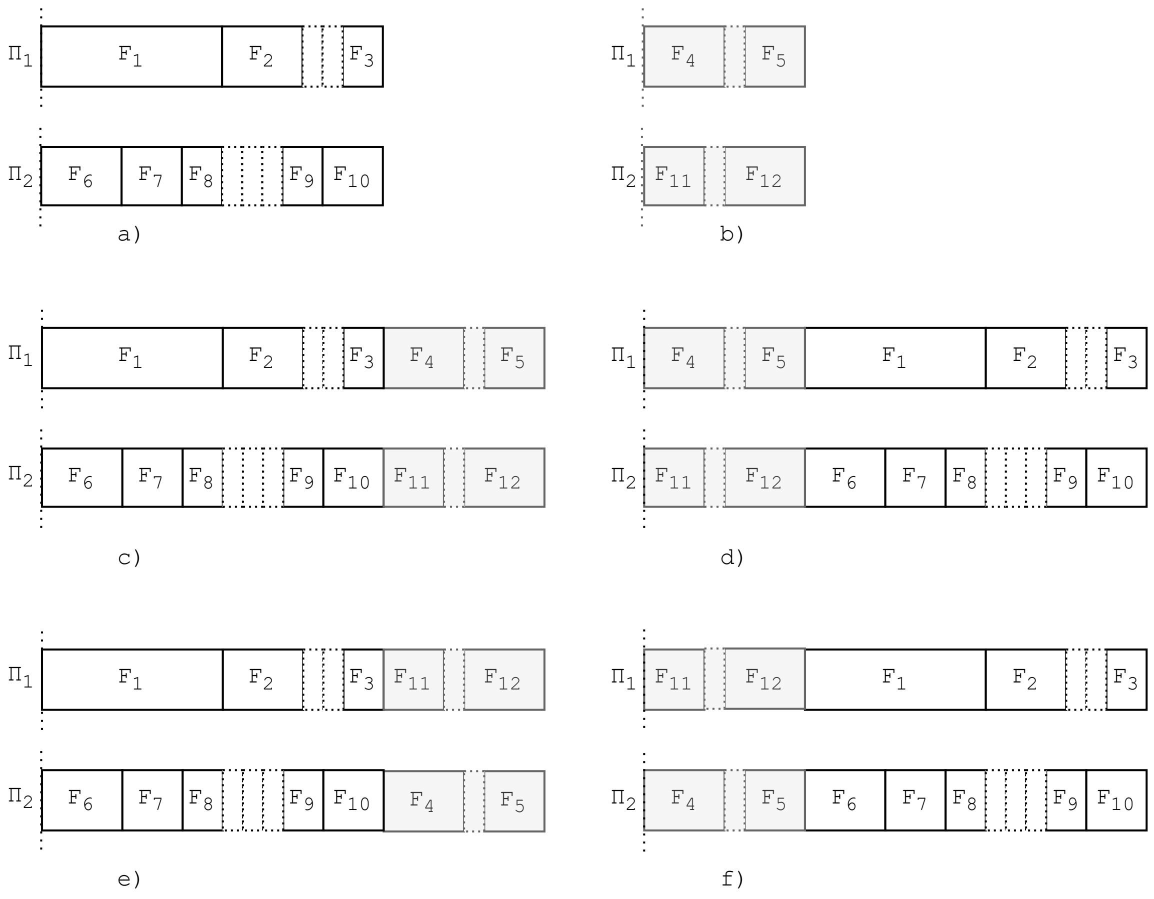
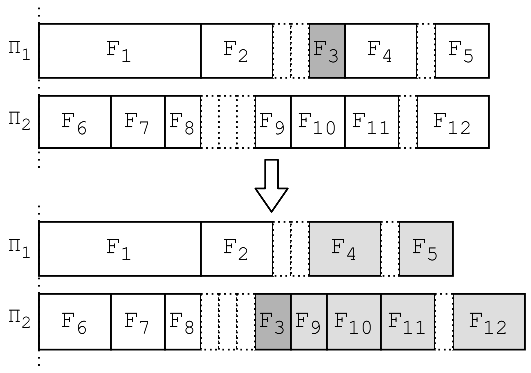
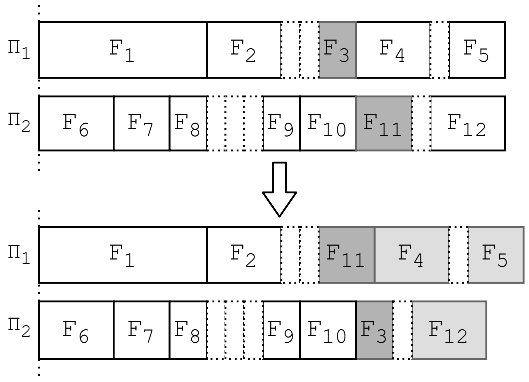


Figure 4.7: Proposed merging constructive method, showing all different ways that two facilities groups can be merged. *a* and *b* are the original groups, while *c*, *d* *e* and *f* are all valid combinations.



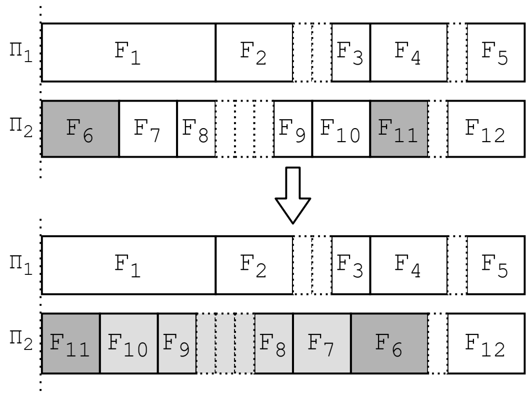
π_1
π_2
F_1 F_2 F_3 F_4 F_5
F_6 F_7 F_8 F_9 F_{10} F_{11} F_{12}

(a) Move one facility between any position in any row.



π_1
π_2
F_1 F_2 F_3 F_4 F_5
F_6 F_7 F_8 F_9 F_{10} F_{11} F_{12}

(b) Swap two facilities between any rows.



π_1
π_2
F_1 F_2 F_3 F_4 F_5
F_6 F_7 F_8 F_9 F_{10} F_{11} F_{12}

(c) *2Opt*: Reverse a fragment of a row.

Figure 4.8: Implemented neighborhoods for the DRFLP.

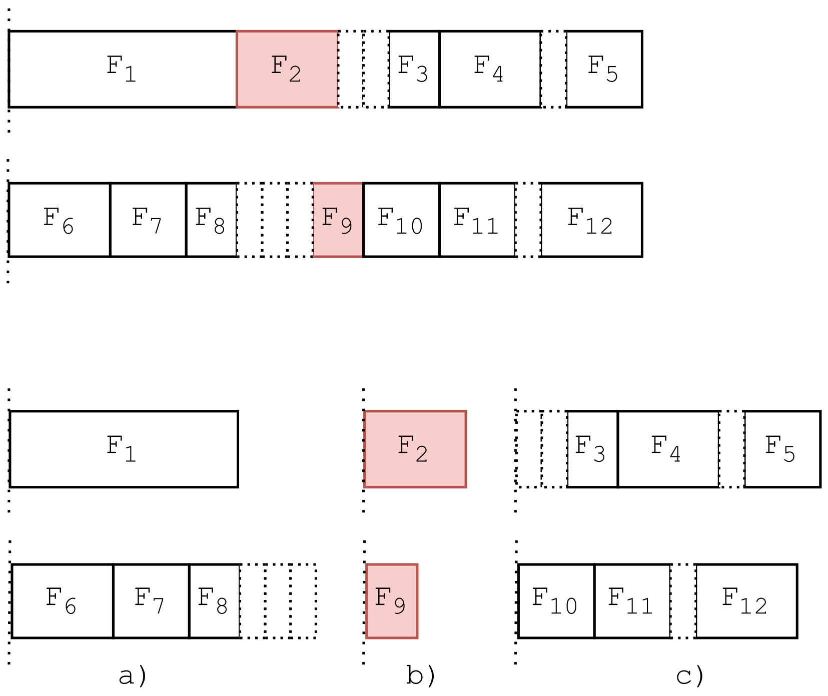


Figure 4.9: Proposed destructive method, based on piece splitting. *a*, *b* and *c* are the resulting pieces after the split.

After an in-depth tuning phase using `irace`, the IG proposal will apply the second constructive to generate the initial solutions, based on inserting facilities in any position, with $\alpha = 0.73$; the splitting destructive with the agglomerating constructive method as reconstructive, with $\alpha = 0.3$; a best improvement local search formed by the swap neighborhood; and finally, and a linear increment strategy for adding dummy facilities in each iteration. For a full detail of the preliminary experiments, the reimplementation of the state of the art and full algorithmic detail of the components, see Chapter 8.

The final comparison against the four existing heuristic algorithms proposed in [1] uses the exact same set of 38 instances. Results are summarized in Table 4.4, where the following metrics are reported for each algorithm: average execution time in seconds T(s), average deviation to best known value (% Dev.) and number of best known values found (# Best). The IG proposal obtains 38 best values, while the previous heuristic methods obtain between 27 and 31. However, due to the small gaps, we can infer that most results are near the optimal value, and therefore bigger instances may be required in order to clearly differentiate the performance of the proposals.

Table 4.5 shows the summarized results using 15 new instances. In this case, the IG proposal clearly outperforms all state of the art heuristic methods, obtaining all known best values in a fraction of the time required by the heuristics.

	H1	H2	H3	H4	IG
T (s)	8575.2	7456.2	9335.7	9246.0	**27.2**
% Dev.	0.01	0.01	0.01	0.01	**0.00**
# Best	31	31	26	27	**38**

Table 4.4: Comparison between the four heuristic methods proposed in [1] against our *ig* proposal. Instance set formed by 38 instances.

	H1'	H2'	H3'	H4'	IG
Time (s)	3600.8	3600.8	3600.8	3600.6	**429.4**
% Dev	0.37	0.47	0.28	0.25	**0.00**
# Best	0	0	0	0	**15**

Table 4.5: Comparison between the reimplementation of the four heuristic methods proposed in [1] against our *ig* proposal. Instance set formed by 15 instances.

4.4 Results for ***AutoConfig***

To wrap up the results chapter, we will use the automatic configuration methodology presented in Section 2.4 to compare the performance of the algorithms presented in the previous sections against an automatically generated configuration, with no prior knowledge or baselines. In order to make a fair comparison, the existing approaches are slightly modified: all existing parallelization code is disabled, and all parameters are standardized and exposed using a unified API. In order to make the experimentation robust, we re-executed the adapted code 30 times.

Moreover, we will use a variant of the DRFLP problem, the SF-DRFLP. The objective is to demonstrate the effectiveness of the approach using at least one implementation not written by the author, and how it can easily be generalized and applied to different problems. As the original code for the SF-DRFLP is written in C++, the code will be rewritten in Java. In order to demonstrate that the rewritten code performs similarly to the original one, we run a preliminary experiment in which we execute both approaches 30 times for all instances. An average difference of 0.02% is obtained between both implementations, which shows minimal behavior differences between both implementations.

The automatic configuration experiment will be run as follows. For each pair of instance and candidate configuration, a runtime limit of 60 seconds is imposed. The objective function to optimize for all problems will be the Area Under Curve (AUC) given by the evolution of the objective function during the algorithm execution. Using this function has multiple advantages: first, it works as is for both maximization and minimization problems, as we want to minimize the area when we are solving a minimization problem and maximize it if not. Moreover, it simplifies comparisons between different algorithm configurations, specially when each algorithm has a different execution time. For more information about anytime optimization and its relationship to algorithm tuning, see [109].

The chosen parameter optimizer is `irace` [21], with a budget of 10.000 executions per 50 parameters. This way, the experiment scales appropriately as the parameter space increases. In order to perform a fair comparison, the best configuration found will be executed 30 times using as time limit the average execution time of the adapted code.

In Table 4.6, we present the summarized results for the three problems detailed previously. For every problem, the following metrics are reported: number of times that the generated configuration reaches the best known value for all instances (#Times reaches *bkv*); number of instances for which the algorithm obtains the best known value at least once (#Instances finds *bkv*); averaged percentage deviation of the best value found by the configuration to the best known value (%Dev Min); and lastly, averaged percentage deviation of all iterations to the best known value (%Dev Avg).

SF-DRFLP	*AutoConfig*	Reimplementation
#Times reaches *bkv*	485 (19.96%)	895 (36.83%)
#Instances finds *bkv*	78 (96.30%)	42 (51.85%)
%Dev Min	0.00	0.01
%Dev Avg	0.45	0.03
BMSSC		
#Times reaches *bkv*	221 (29.47%)	206 (27.47%)
#Instances finds *bkv*	18 (72.00%)	17 (68.00%)
%Dev Min	0.03	0.20
%Dev Avg	0.46	0.46
VRPOD		
#Times reaches *bkv*	3816 (30.29%)	2373 (18.83%)
#Instances finds *bkv*	372 (88.57%)	132 (31.43%)
%Dev Min	1.55	0.68
%Dev Avg	5.06	1.91

Table 4.6: Comparison between the automatically generated configurations for each of the proposed problems and the adapted code. *bkv* are the initials of best known value.

We can observe that the automatically generated algorithm finds the best known value for most instances, improving the results of the existing proposal, for the three problems. Moreover, it reaches the best value more times for both the VRPOD (30% vs 19%) and the BMSSC (72% vs 68%), while having a lower value of 20% for the SF-DRFLP. The explanation for the vast difference in this problem between both metrics can be easily explained by analyzing the detailed results for the SF-DRFLP. In most instances, the original code either finds the best value in all iterations, or none of them. The deviations for the SF-DRFLP displays the same behavior.

Analyzing the minimum and average deviations, we can extract additional information from Table 4.6. First, for the SF-DRFLP, both minimum deviations are very low, while the average deviation is slightly higher for the *AutoConfig* proposal (0.45%). In the case of the BMSSC, the *AutoConfig* proposal obtains a lower deviation minimum deviation (0.03%) to the best known values, and an equal average of 0.46%.

Lastly, for the VRPOD, the *AutoConfig* reaches the best known value around 30% of the time, having the best value for 89% of instances, while the reimplementation obtains 19% and 31% respectively. However, both deviations are worse for the *AutoConfig*. This can be explained due to the fact that for about 1% percentage of the VRPOD instances, the minimum deviation is greater than 10%, and around 6% for the average deviation. The root cause is probably that this uncommon instance type, or any instance with similar characteristics, was not seen enough during the *AutoConfig* tuning step. Moreover, the big difference in performance according to the number of best values reached, may be explained by the instance types. Instances in the VRPOD can be divided according to the compensation scheme used to pay the occasional drivers. Analyzing the detailed results, we have observed that the *AutoConfig* proposal obtains good results for both compensation scheme types available in the state of the art, while the reimplementation performs well for the first compensation scheme but falls behind in the second type. Due to this, we may conclude that the *AutoConfig* generalizes better the instances properties, and therefore it is much more likely to win under new instance sets.

To sum up, the automatic configuration methodology is able to find algorithms with equal or greater performance than traditional manual tuning, in an automatic and reproducible way.

Chapter 5

Conclusions and future work

In this chapter, the contributions made during the thesis development are summarized, along their impact, and the conclusions that may be extracted from them. Moreover, future liens of work and possible further improvements to the presented ideas are presented.

5.1 General conclusions

In Section 1.3, we defined the initial hypothesis and the main objective of the thesis as developing and validating a new methodology that ensures consistent reproducibility during the whole research lifecycle, facilitating, among others, three key aspects: automatic instance selection, automatic generation and configuration of proposals and artifact generation. As demonstrated in the results presented in Chapter 4, the initial hypothesis has been verified, validating that an automatic experiment design, named *AutoConfig*, can match and improve manually tuned proposals. To do so, we have used three problems from different families with different solution structures: a routing problem (VRPOD), a clustering problem (BMSSC) and lastly a facility layout problem (SF-DRFLP).

One limitation of the methodology is that it cannot guarantee the optimality of the generated algorithms, or in other words, there may exist different combinations of parameter values which obtain better results than the proposed algorithm configurations. However, our objective is not guaranteeing the optimality of the automatically generated proposals, but ensuring that we can procedurally test, build and validate their performance, ensuring their reproducibility, while improving the results obtained if compared against a manual tuning approach.

All artifacts, full tables, generated figures and the scripts used to generate them, complete source code, and executable artifacts for each problem are available at the repositories specified in Table 5.1. Specifically, as proposed in the methodology, we provide public containers to ease the execution of the artifacts, a live version of the source code available at the GitHub repositories and a permanently archived version in

Zenodo at the date of this thesis. Moreover, all instance data and results analysis are classified in packages according to the problem they belong and archived in Zenodo.

Problem	Live Code	Archived Artifacts DOI
BMSSC	rmartinsanta/ac-BMSSC	10.5281/zenodo.7774638
VRPOD	rmartinsanta/ac-VRPOD	10.5281/zenodo.7774831
SF-DRFLP	rmartinsanta/ac-SFDRFLP	10.5281/zenodo.7774833

Table 5.1: References to all published artifacts related to this thesis development.

5.2 Contributions summary and future work

In this thesis, we propose a new methodology to tackle two hot topics in metaheuristic research, research reproducibility and robust decision automation, specially applicable to the instance selection and the algorithm tuning phases of research. We do not limit ourselves to a theoretical proposal, and a reference open source implementation under a permissive license is provided, called *Mork*, which stands for Metaheuristic Optimization FramewoRK. Its latest version is available at [1]. This reference implementation is used in Section 4.4 to automatically build and benchmark the different provided algorithmic components, improving the results obtained by using a manual tuning approach. The methodology implementation has been iteratively implemented and improved for the proposed problems, and thanks to it, we have improved the reproducibility of the proposals, while simultaneously reducing the workload of the researcher. Thanks to the decision automation, experiments can be generated by the framework, simplifying the experimentation workflow, using a consistent and easy-to-understand approach.

Moreover, the state of the art for the three problems tackled during the thesis has been improved applying the methodology. All relevant findings and results have been published in journals of the area. During the thesis development, we have achieved 5 JCR publications, four of them in journals ranked Q1, and the remaining one in a journal ranked Q2. Additionally, we have published 2 LNCS ranked Q2 in SJR, a book chapter about metaheuristics, and presented 8 works in top national and international congresses of the area, such as ICVNS, MIC and CAEPIA.

One indirect contribution that we have seen emerge is that, by proposing a reference implementation, with common interfaces and detailed examples, multiple researchers have started sharing components and algorithm implementations between different research projects. Thanks to the ease of component reuse, and their flexibility, the total work required for creating new strategies is further reduced. Not only that, there are currently multiple research works published or in process of being published using *Mork*, examples of published works are [110] and [111]. As such, we expect the list of related works to increase over time.

[1]`https://github.com/mork-optimization/mork`

Lastly, there are several areas that require further research. Depending on the problem, and the instance data available in previous works, the number of instances ranges from single digit to thousands. Currently, all instances are passed to `irace` during the parameter optimization phase. It would be interesting to measure the influence of the different instance selection strategies, to see how it affects both the runtime of the *AutoConfig* experiment and the results obtained. Examples of strategies to use are, doing an automatic previous selection of instances, to guarantee that `irace` has seen at least once each type of instance; using all instances as it currently does or allowing the user to select a preliminary experiment set. Automatically selecting instances is not trivial, as defining a type of instances is very context dependent, and usually requires specific data provided by the user.

Another idea that may be worth exploring is integrating other parameter tuning engines, for example based on Bayesian optimization [112], or either random or grid searches [113], and allow the user to select which one to use. As it is currently designed, the parameter tuning engine is a black box component that may be easily replaced by another implementation if needed.

Moreover, there is a limited number of components available. We will keep adding generic components that can be used as is. For example, we are currently working on adding population based methods. Additionally, we are currently modifying the reference implementation to allow users to trivially distribute the experiment workload to computing clusters of any size, without requiring any code change to the existing components. Thanks to this, we mitigate the possible relatively long time required for the tuning step in the proposed methodology, reducing the required total execution time from days to hours.

Another aspect that may be improved is the explainability of the generated configurations. We would like to generate a report after the automatic configuration that details why some components are preferred to another, why some configurations are discarded, and the best range of values found for parameters depending on the context. This way, the user may manually create new approaches with the previous knowledge of expected performance, saving experimentation time. Moreover, we are implementing the inverse, that is, allowing the user to provide knowledge to the automatic experimentation engine. Results may be improved if the users provide several existing well performing configurations, so the automatic configuration engine does not start from scratch for each experiment, and can discard poorly performing approaches sooner.

Finally, while the current design may work with small adaptations to multiobjective problems, its application has not been studied in depth. A future work will explore multiobjective optimization problems from different families and document any adaptation necessary or peculiarities found.

Part II

Publications: published, accepted, and submitted papers

Contents

Chapter 6

Strategic oscillation for the balanced minimum sum-of-squares clustering problem

Title	Strategic oscillation for the balanced minimum sum-of-squares clustering problem
Authors	Raúl Martín-Santamaría, Jesús Sánchez-Oro, Sergio Pérez-Peló and Abraham Duarte
Publication date	2021
Journal	Information Sciences (Q1)
Publisher	Science Direct
ISBN/ISSN	0020-0255
Impact Factor	8.233 (2021)
Rank by Impact Factor	16/164 (Q1, Computer Science, Information Systems)
DOI	`https://doi.org/10.1016/j.ins.2021.11.048`

Information Sciences 585 (2022) 529–542

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Strategic oscillation for the balanced minimum sum-of-squares clustering problem

R. Martín-Santamaría [a], J. Sánchez-Oro [a,*], S. Pérez-Peló [a], A. Duarte [a]

[a] Dept. Computer Sciences, Universidad Rey Juan Carlos, Spain

A R T I C L E I N F O

Article history:
Received 6 March 2020
Received in revised form 1 June 2021
Accepted 14 November 2021
Available online 26 November 2021

Keywords:
Balanced clustering
Metaheuristics
Strategic oscillation
GRASP
Infeasibility

A B S T R A C T

In the age of connectivity, every person is constantly producing large amounts of data every minute: social networks, information about trips, work connections, etc. These data will only become useful information if we are able to analyze and extract the most relevant features from it, which depends on the field of analysis. This task is usually performed by clustering data into similar groups with the aim of finding similarities and differences among them. However, the vast amount of data available makes traditional analysis obsolete for real-life datasets. This paper addresses the problem of dividing a set of elements into a predefined number of equally-sized clusters. In order to do so, we propose a Strategic Oscillation approach combined with a Greedy Randomized Adaptive Search Procedure. The computational experiments section firstly tunes the parameters of the algorithm and studies the influence of the proposed strategies. Then, the best variant is compared with the current state-of-the-art method over the same set of instances. The obtained results show the superiority of the proposal using two different clustering metrics: MSE (Mean Square Error) and Davies-Bouldin index.

1. Introduction

The large amount of data available in several fields of research like economics or biology has made traditional analysis techniques impractical for real-life problems [35]. In fact, data used in most of these areas grows exponentially everyday. Therefore, the design of high-quality and high-performance algorithms has become a research field of interest for data analysts.

In order to get relevant information when using huge volume of data, two main approaches are typically followed: classification and clustering [20]. In the former, there is a complete previous knowledge of the available information (i.e., the group to which each element belongs to is known beforehand). Then, data analysis techniques can take advantage of this information in a supervised way. On the contrary, clustering techniques use this information in an unsupervised way, since it is not available *a priori*. The design of effective procedures for clustering is harder as the actual group for each element is not known [39].

Clustering problems consist in splitting data into groups (also known as clusters), which contain elements that share some specific features. In other words, it is expected that if two elements belong to the same cluster, then it is because they

* Corresponding author.
E-mail addresses: raul.martin@urjc.es (R. Martín-Santamaría), jesus.sanchezoro@urjc.es (J. Sánchez-Oro), sergio.perez.pelo@urjc.es (S. Pérez-Peló), abraham.duarte@urjc.es (A. Duarte).

https://doi.org/10.1016/j.ins.2021.11.048
0020-0255/© 2021 Elsevier Inc. All rights reserved.

are related to each other by means of some specific features. Symmetrically, if two elements are in different clusters, it is because they are barely related. Therefore, clustering algorithms are designed to split a given dataset into several subsets maximizing the similarity of elements in the same subset, while minimizing the similarities between elements in different subsets. It is important to remark that the similarity metric used totally depends on the dataset and the scope of the clustering.

The minimum sum-of-squares clustering problem (MSSC) presents several practical applications in a wide variety of areas (biology, biometry, psychology, marketing, etc.), and it is also a useful technique to improve the performance of other techniques like pattern recognition, data mining, or image processing, among others [36]. This problem has been proven to be $\mathcal{NP}$-hard [24], even when considering only two clusters in the Euclidean space [2].

MSSC considers that the number of clusters k is known *a priori*. The objective of MSSC is to assign a set of $n$points $P = \{p_1, p_2, \ldots, p_n\}$ located in an s-dimensional Euclidean space $\mathbb{R}^s$ to a cluster $K_i \in \{K_1, K_2, \ldots, K_k\}$. MSSC then tries to find the assignment S of points to clusters with the minimum sum-of-squares distances from each point p_j of the cluster K_i to its corresponding centroid $\bar{p}_i$. Given a cluster K_i, its centroid is defined as the point whose distance to the other points of the cluster is minimum. More formally, the quality of the partition S, denoted as $f(S)$ can be evaluated as:

$$f(S) = \sum_{j=1}^{n}\sum_{i=1}^{k} x_{ij}||p_j - \bar{p}_i||^2$$

where x_{ij} is a binary variable (with $1 \leqslant i \leqslant k$ and $1 \leqslant j \leqslant n$) that takes on the value 1 if point p_j is assigned to cluster K_i; otherwise, $x_{ij} = 0$. Naturally, this variable satisfies that $\sum_{i=1}^{k} x_{ij} = 1$ for $1 \leqslant j \leqslant n$.

The evaluation of the objective function can be improved by leveraging the results derived from the Huygens' theorem [13], as stated in [9]. Specifically, evaluating the sum-of-squared distances from all points of a given cluster to its centroid is equivalent to evaluating the sum-of-squared distances between each pair of points in the cluster divided by its cardinality. It can be formally defined as:

$$f(S) = \sum_{i=1}^{k}\sum_{j=1}^{n-1}\sum_{l=j+1}^{n} x_{ijl}\frac{||p_j - p_l||^2}{|K_i|}$$

where x_{ijl} is a binary variable (with $1 \leqslant i \leqslant k$ and $1 \leqslant j < l \leqslant n$) that takes on the value 1 if points p_j, p_l are assigned to cluster K_i; otherwise, $x_{ijl} = 0$.

It is worth mentioning that the use of the Huygens' theorem [13] allows us to design considerably efficient algorithms. Specifically, we compute the distances between each pair of elements only once (even before executing the algorithm) storing them in a matrix. Then, looking up the distance between two elements can be performed in $O(1)$ since it only requires an access to the distance matrix. Additionally, this approach makes the algorithms independent of the instance dimensionality. See [4,21] for further details about the issues related with the dimensionality course.

In this paper, we focus on the variant with the cardinality constraint, which is called Balanced MSSC (BMSSC) problem. Specifically, there will be $n \bmod k$ clusters of size $\lceil n/k \rceil$, and $k - (n \bmod k)$ clusters of size $\lfloor n/k \rfloor$. BMSSC has also been proven to be $\mathcal{NP}$-hard for $n/k \geqslant 3$. See [28] for further details.

Fig. 1 shows an example of two possible clustering solutions for $k = 2$, where we consider the same set of points. On the one hand, Fig. 1.a depicts a solution S_1 conformed with $K_1 = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{F}, \mathtt{H}\}$ and $K_2 = \{\mathtt{D}, \mathtt{G}, \mathtt{E}, \mathtt{I}\}$, resulting in an objective function value of $f(S_1) = 245$. On the other hand, the solution S_2 presented in Fig. 1.b contains clusters $K_1 = \{\mathtt{A}, \mathtt{C}, \mathtt{D}, \mathtt{G}, \mathtt{I}\}$ and $K_2 = \{\mathtt{B}, \mathtt{E}, \mathtt{F}, \mathtt{H}\}$, with an objective function value of $f(S_2) = 353$. Analyzing these values, we can conclude that S_1 is better than S_2 since the elements contained in each cluster are more similar among them (regarding the computation of the objective function).

In this paper, we propose a Greedy Randomized Adaptive Search Procedure (GRASP) [15] combined with Strategic Oscillation (SO) [16]. Fig. 2 shows the complete scheme of the proposed algorithm, where we can identify two main blocks: GRASP and SO. The first one performs a predefined number of constructions followed by a local search method. Then, the best solution found is further improved in the SO block by alternatively considering feasible and unfeasible solutions. The SO block finally returns the best solution found during the search.

This procedure presents a remarkable performance in both, quality and computing time, as we will show in the computational experience. The remaining of the paper is organized as follows: Section 3 presents the metaheuristic algorithm developed for the BMSSC, Section 4 describes a post-processing method in order to improve the quality of the obtained solutions, Section 5 thoroughly describes the experiments performed to test our proposal and, finally, Section 7 draws some conclusions over the problem and the proposed algorithms.

2. Literature review

Most of the clustering problems have been proven to be $\mathcal{NP}$-hard [17]. Therefore, the majority of the algorithms found in the related literature are approximate procedures, where the main goal is to find a high quality partition in reasonable computing times. However, these algorithms are not able to guarantee the optimality of the provided solution. Clustering algo-

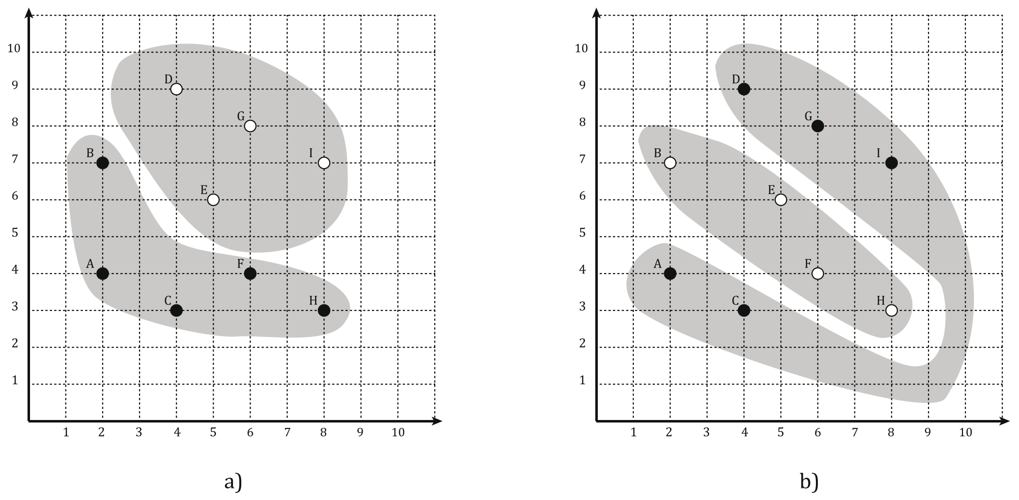


Fig. 1. Two possible clustering solutions for a given set of points: (a) {A,B,C,F,H} and {D,G,E,I}, and (b) {A,C,D,G,I} and {B,E,F,H}.

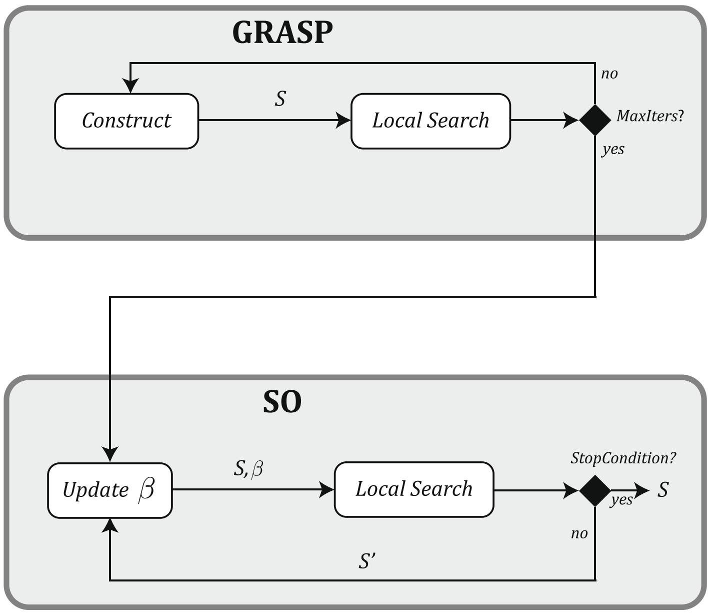


Fig. 2. Complete scheme of the proposed algorithm.

rithms can be classified in: partitional clustering [29], in which data must be divided into disjoint sets and each element is assigned to one set; hierarchical clustering [5], where clusters are created following a top-down or bottom-up approach; density-based clustering [27], where elements are grouped following a density function; or grid-based clustering [19], in which data is divided into grids with different granularity level. We refer the reader to [34] for a detailed taxonomy of basic and advanced clustering techniques.

In this paper, we focus on partitional clustering whose objective is to generate non-overlapping subsets of elements where each element is assigned to exactly one cluster. The concept of similarity can be defined in a large variety of criteria, but usually they do not coincide. However, there is a common criterion when considering the clustering of elements that can be located in an Euclidean space. Specifically, it is widely accepted that minimizing the sum of squares between elements in the same cluster is a good criterion for clustering analysis. Furthermore, it is equivalent to maximizing the sum of squares between elements in different clusters, resulting in a criterion for increasing both similarity in the same cluster and separation among different clusters [36].

The *k*-means procedure has been widely used mainly due to its simplicity and its computational efficiency [31,37]. However, it totally relies on the initial random centroid selection. Therefore, if that selection results in a bad initial set of centroids, the method will not be able to obtain a good partition. A new method, named *k*-means++ [3], was designed to mitigate this behavior and perform a better initial centroid selection. Specifically, it consists in selecting the most diverse centroid; that is, the one with the smallest similarity measure among them. This idea reduces the harmful effect that a random centroid selection can produce in the performance of the traditional k-means algorithm. Recently, there have been another approaches proposed to even reduce this effect. In particular, the power k-means algorithm is proposed to avoid poor local minima by annealing through a family of smoother surfaces [38]. This algorithm is further improved in [6] when dealing with high dimensions. Finally, in [8] is described a different approach, denoted as convex clustering, where a penalty function is introduced to guarantee the convexity of the derived problem.

The classical Minimum Sum-of-Squares Clustering (MSSC) problem does not have any constraint about the number of elements that can be assigned to each cluster. However, several problems require generating clusters of similar sizes. Desrosiers et al. [11] proposed a Variable Neighborhood Search algorithm for dividing students from school and universities in teams, considering that each team must provide a good representation of the population. They propose two different functions for evaluating the balance among teams and test the efficiency of the algorithm over actual data from an MBA program. This problem also has applications in Very Large Scale Integration (VLSI) design. Specifically, Hagen et al. [18] indicates that the second smallest eigenvalue of a matrix derived from the corresponding netlist provides a good approximation to the optimal ratio cut partition cost. As is stated in this paper, balance clustering problems are equivalent to the second eigenvector computation. Consequently, an effective and efficient method for the former is useful for the later. Finally, Su et al. [33] present an algorithm for balancing tenant placement in cloud computing, with the aim of improving the performance and maximizing the resource utilization of complex multi-tenant architectures.

First attempts to solve the Balanced Minimum Sum-of-Squares Clustering (BMSSC) problem consider the well-known k-means procedure [32]. This algorithm is a classical clustering method that consists of two main steps: it firstly selects the elements that will become the centroids of the clusters and then assigns each one of the remaining elements to their nearest centroid. We refer the reader to [22] for an efficient implementation of the method.

The best heuristic algorithm identified in the related literature is presented in [9]. The procedure is based on the Variable Neighborhood Search methodology, which relies on the idea of combining stochastic and deterministic changes of neighborhood to escape from local optimality. Therefore, we include this algorithm in the computational testing. In order to complement the experimentation, we include k-means [3] with the Hungarian algorithm [23] to satisfy the balance constraint among clusters. Finally, we include two of the most performing procedures for the MSSC problem. Specifically, the hierarchical Clustering with Optimal Transport (HCOT) method [5] and the Continuous GRASP [29]. In both methods, a post-processing method is executed to guarantee the balance constraint (i.e., oversized clusters are repaired by reassigning their elements to the best available cluster).

3. Greedy randomized adaptive search procedure

The term Greedy Randomized Adaptive Search Procedure (GRASP) refers to a metaheuristic algorithm originally introduced in the late 1980s [14] but it was not formally defined until 1994 [15]. GRASP is an iterative algorithm where each iteration can be divided into two stages: generating an initial solution and then locally improving it. The first phase starts from an empty solution building the Candidate List (CL) of elements to be added to the solution under construction. The first element is usually selected at random from the CL. The remaining elements to be included in the solution are selected following a greedy criterion. Specifically, a Restricted Candidate List (RCL) is conformed with those elements that surpass the greedy criterion established. Then, the next element to be added is selected at random from the RCL. This random selection allows GRASP methodology to explore diverse regions of the search space, thus increasing the possibility of finding better solutions.

The random part of the initial solution construction is able to generate diverse solutions, but it is not designed to find a local optimum with respect to the constructed solution. Therefore, a local improvement method is required in order to find a local optimum with respect to the constructed solution. The versatility of the GRASP methodology allows us to use different algorithms in this stage, from traditional local search methods to more complex implementations such as hybridizations with other metaheuristics that has lead to successful research: Tabu Search [26], Path Relinking [12], Ejection Chains [30], among others.

Finally, this two stages are iteratively repeated until a stopping criterion is reached, which is usually a maximum number of generated solutions or a maximum allowed running time. The method returns the best solution found during the search. The remaining of this section is devoted to presenting the specific design of the GRASP algorithm for the BMSSC.

3.1. Initialization

Classical GRASP algorithms usually select the first element to be added to the solution under construction at random. However, as stated in Section 1, the random selection of the initial elements in a clustering problem can determine the quality of the obtained results. Therefore, we propose a new initialization criterion which tries to guide the initial solution to promising regions of the search space by inserting a single vertex in each cluster. The method starts by selecting the first element at random and inserting it in the first cluster. Then, the distance from the remaining elements to the one already selected is evaluated, selecting the element that presents the largest one (i.e., the one that is furthest from the element already clustered).

Once two elements have been assigned to two different clusters, the next element to be inserted should be far away from both elements already assigned. For this purpose, we define the distance from an element p to a given cluster K_i as follows:

$$d(p, K_i) = \sum_{p_j \in K_i} ||p - p_j||^2$$

where K_i is the set of points $p_j \in P$ that have been assigned to cluster i.

Then, the distance from an element $p$to a given solution $S = \{K_1, K_2, \dots, K_k\}$ can be defined as:

$$d(p, S) = \min_{K_i \in S} d(p, K_i)$$

Then, the next element to be added, $p^{\star}$, will be the one with the largest distance to the already clustered elements. More formally:

$$p^{\star} = \underset{p \in P \setminus S}{\text{argmax}}\, d(p, S)$$

This criterion allows us to insert, in each cluster, the furthest element with respect to the already clustered elements, thus reducing the similarity of the elements in different clusters and, therefore, increasing the similarity between elements in the same cluster.

The initialization stage ends when each cluster has exactly one element in it, using this partial solution as input for the constructive method.

3.2. Constructive method

The constructive algorithm proposed in this work, named *JoinClosest*, follows a traditional GRASP approach where each element is added to the cluster that minimizes the value of the objective function. *JoinClosest*, as a GRASP constructive method, requires from a greedy function that evaluates the relevance of adding an element in a given step of the construction.

The greedy function for each element is evaluated as the increase in the objective function value if the element is inserted in the closest cluster. It is worth mentioning that only the clusters which are not completed yet are considered in this step, in order to maintain the feasibility of the solution. More formally,

$$g(v, S) = \min_{1 \leqslant i \leqslant k} \sum_{p \in K_i} d(p, v)$$

```
Algorithm 1 $JoinClosest(P, S = \{K_1, K_2, \dots K_k\}, \alpha)$
1: $CL \leftarrow P \setminus S$
2: **while** $CL \neq \emptyset$ **do**
3:   $g_{\min} \leftarrow \min_{c \in CL} g(c, S)$
4:   $g_{\max} \leftarrow \max_{c \in CL} g(c, S)$
5:   $\mu \leftarrow g_{\min} + \alpha \cdot (g_{\max} - g_{\min})$
6:   $RCL \leftarrow \{c \in CL : g(c, S) \leqslant \mu\}$
7:   $c^{\star} \leftarrow Random(RCL)$
8:   $CL \leftarrow CL \setminus \{c^{\star}\}$
9:   $K^{\star} \leftarrow \underset{K_i \in S}{\text{argmin}}\, d(c^{\star}, K_i)$
10:  $K^{\star} \leftarrow K^{\star} \cup \{c^{\star}\}$
11: **end while**
12: **return** $S$
```

Algorithm 1 depicts the pseudocode of the *JoinClosest* constructive procedure. The algorithm receives three input parameters: P, the set of points that needs to be clustered; S, the set of clusters created in the initialization step; and α, a parameter that controls the greediness/randomness of the method (discussed later).

The method starts by creating the Candidate List (CL) with the set of elements in Pthat has not been assigned to any cluster in the initialization phase (step 1). Then, it iterates until all the elements have been clustered (steps 2–11), adding a new element to a cluster in each iteration.

Specifically, *JoinClosest* evaluates the minimum and maximum value for the greedy function previously defined (steps 3–4) and then evaluates a threshold μ (step 5) that depends on the value of the parameter $\alpha \in [0, 1]$ whose function is to limit the elements that are allowed to enter the Restricted Candidate List (RCL). On the one hand, $\alpha = 0$ indicates that only the elements with the best greedy function value are selected, which results in a totally greedy algorithm. On the other hand, $\alpha = 1$ considers all the elements in the CL to be added to the RCL, which results in a totally random algorithm. Therefore, parameter α is able to control the greediness/randomness of the constructive procedure.

Once the *RCL* is created with the elements whose objective function value is smaller than the threshold (step 6), an element $c^{\star}$ is selected at random from the *RCL* (step 7). Then, the algorithm selects the closest cluster $K^{\star}$ (step 9) and inserts the element in the cluster (step 10). The method ends when all the elements have been assigned to a cluster, returning the solution created.

3.3. Local optimization

The solution created in the construction phase tries to find a balance between diversification and intensification in order to explore a wider portion of the search space. However, this behavior complicates finding a local optimum in the construction stage. The second phase of a GRASP procedure consists of an improvement strategy that is able to find a local optimum of the initial solution with respect to a given neighborhood.

In this work, we consider a neighborhood based on interchanging two elements of different clusters. More formally, given a solution

$$S = \{K_1, \ldots, K_a, \ldots, K_i, \ldots, K_k\}$$

the interchange of elements $p_j \in K_a$ and $p_l \in K_i$, produces a new solution $S\prime = \{K_1, \ldots, K_a \setminus \{p_j\} \cup \{p_l\}, \ldots, K_i \setminus \{p_l\} \cup \{p_j\}, \ldots, K_k\}$. For the sake of clarity, we represent this move as $S\prime \leftarrow move(S, p_j, K_i)$.

We propose a local search method for the BMSSC based on this neighborhood. It evaluates the interchange of every pair of elements in the solution, executing in each step the first interchange that results in a better solution (first improvement, FI). We additionally propose a different approach in which the interchange performed is not the first element that leads to a better solution but the one that leads to the best solution in the neighborhood (best improvement, BI). The performance of both local search methods will be later discussed in Section 5.

4. Strategic oscillation

Reaching a better solution during the search might prove difficult in some cases, since the constraint on the size of each cluster in the BMSSC problem limits the number of available movements that can be performed. Specifically, in order to maintain the same size in each cluster it is only possible to perform symmetrical movements (mostly based on interchanges).

With the aim of increasing the portion of the search space explored, we propose considering unfeasible solutions during the search that can eventually lead the algorithm to better feasible solutions.

Strategic oscillation (SO) is a methodology originally proposed for being used in combination with Tabu Search [16]. It is based on allowing the algorithm to surpass the boundaries of its search space, usually by consider the exploration of unfeasible solutions. When the search gets stuck in a deep basin of attraction, SO modifies the rules of the search, allowing the algorithm to continue its exploration considering the set of unfeasible solutions. Every time a promising unfeasible solution is reached, the algorithm must repair it in order to transform it into a feasible solution that would eventually be a high-quality one.

The first step in SO is the definition of the boundary to be surpassed. In the context of BMSSC, we consider the increment of the size of each cluster. This modification is performed by increasing each cluster size by a percentage defined by a parameter $\beta \in [0, 1]$ that controls how far the explored solutions are from being feasible. Specifically, given a cluster K_i, if the original cluster size is $|K_i|$, considering the relaxation of the feasibility, the new cluster size is now limited by $|K_i| \cdot (1 + \beta)$. This relaxation allows the algorithm to include more points in each cluster, which may lead the procedure to find better solutions that can be later repaired. A search algorithm that considers small values of β will explore solutions that are almost feasible, while considering larger values will explore rather unfeasible solutions.

SO allows us to define a new neighborhood to be explored, which consists in moving a given element from one cluster to another. Notice that this movement cannot be considered in the feasible solution space since any move violates the size constraint. Given a point p_j that belongs to cluster K_a which is moved to cluster K_i in a certain solution $S = \{K_1, \ldots, K_a, \ldots, K_i, \ldots, K_k\}$, the movement generates a new solution $S\prime = \{K_1, \ldots, K_a \setminus \{p_j\}, \ldots, K_i \cup \{p_j\}, \ldots, K_k\}$.

The SO methodology can be divided into two phases: the first one is a local search devoted to exploring the unfeasible region while the second one tries to repair every promising solution.

The local search phase considers the *move* neighborhood previously defined as follows. The method evaluates the move of each element p_j (with $1 \leqslant j \leqslant n$) of the solution $S = \{K_1, K_2, \ldots, K_k\}$ to every cluster K_i (with $1 \leqslant i \leqslant k$). Notice that, in the context of SO, the size of each cluster K_i is now limited by $|K_i| \cdot (1 + \beta)$. The method then selects the cluster $K^{\star}$ that minimizes the objective function value f. More formally,

$$K^{\star} = \underset{1 \leqslant i \leqslant k; 1 \leqslant j \leqslant n}{\operatorname{argmin}} f\left(move(S, p_j, K_i)\right)$$

It is worth mentioning that the local search moves each element to the cluster that minimizes the value of the objective function, resulting in a best improvement method.

The repair phase is applied to every unfeasible solution whose objective function value outperforms the best solution found so far. This phase is intended to reduce the number of elements of each oversized cluster as follows. For each element p_j belonging to an oversized cluster, the method finds the cluster K_i that minimizes the value of the objective function of those whose size satisfies the original size constraint. Then, it applies the operation $move(S, p_j, K_i)$ in order to insert p_j in cluster K_i. The method ends when every cluster satisfies the original size constraint, returning the best solution found during the search.

5. Computational experiments

This section is intended to evaluate the quality of the proposed algorithms and compare it with the best previous approach. All algorithms have been implemented using Java 8 and the experiments were conducted in an Intel Core i5-4210U 1.7 GHz and 8 GB RAM. The source code and the full experimental results are available at the following URL: https://grafo.etsii.urjc.es/BMSSC.

In order to have a fair comparison, we have considered the set of 16 instances used in the best previous work found in the literature. We have additionally incorporated 9 instances to have a larger benchmark. All these instances have been taken from the machine learning repository of the University of California[1]. Table 1 shows, for each single instance, the number of points (n), dimensions (s), and clusters (k). Notice that we first show the original instances and then the new instances, separated by an horizontal line. It is worth mentioning that all instances come from real data. In particular, benchmarks consider information derived from different real world scenarios: breast cancer, phone sensors, water treatment, among others. We refer the reader to [9] to find a deep analysis and description of these instances.

We divide the experiments performed into two different subsets: preliminary experimentation and final experimentation. The former refers to those experiments designed to find the best parameters for the proposed algorithms and to evaluate the relevance of the proposed strategies (constructive method, local search procedure, and Strategic Oscillation algorithm), while the aim of the latter is to perform a comparison between the best algorithm designed and the best previous method found in the state of the art. Notice that the proposed algorithm requires just one run to obtain the presented results.

All the experiments report the following metrics: Dev. (%), the average deviation with respect to the best solution found in the experiment; #Best, the number of times that an algorithm reaches the best solution; and Time (s), the average computing time in seconds.

5.1. Preliminary experimentation

This section is intended to find the best values for the input parameters of the proposed algorithms. Specifically, the algorithms proposed require to find the best value for α and β parameters (constructive procedure and Strategic Oscillation, respectively), as well as selecting the best local search method. We have selected a subset of 4 representative instances (Vehicle, Yeast, Multiple Features, and Image Segmentation) for tuning the parameters of the algorithm in order to avoid overfitting.

The first experiment is devoted to evaluating the performance of constructive method *JoinClosest* when varying the α parameter value. In particular, we have considered $\alpha = \{0.25, 0.50, 0.75, RND\}$, where *RND* indicates that the value is selected at random in the range 0–1 in each construction.

Table 2 shows the effect of the α parameter in the performance of *JoinClosest* when constructing 100 solutions. In particular, the best results are achieved with *JoinClosest*(0.25), which is the closest variant to a totally greedy algorithm. It is worth mentioning that a pure greedy configuration of this constructive method produces worse results than the constructive procedure configured with $\alpha = 0.25$. We can then conclude from this experiment that the larger the randomness, the lower the quality of the constructive procedure. Notice that these results are in line with those presented in [3], which indicates that a random initialization in the well-known k-means algorithm usually produces worse quality solutions than those obtained with the greedy initialization of the k-means++ version.

The main aim of the second experiment is to analyze the ability of each proposed improvement method for finding a local optimum starting the search from each constructed solution. Specifically, we combine the best constructive method (i.e., $\alpha = 0.25$) with the two local search methods proposed in Section 3.3: First Improvement (FI) and Best Improvement (BI).

Table 3 compares the results obtained when combining the best variant of the constructive procedure coupled with both local search algorithms, resulting into two GRASP variants: *JoinClosest*(0.25) + FI and *JoinClosest*(0.25) + BI. We also include the results obtained with *JoinClosest* isolated, in order to analyze the contribution of the improvement strategies. Notice that the results reported in Table 3 are averaged over the 4 instances used in the preliminary experiments. As can be seen in this table, both local search approaches obtain similar results in terms of quality, which can be seen in the average deviation with respect to best found value (0.24% and 0.07%, respectively) and number of best solutions found (both methods matches 2 best solutions). It is also important to remark that avoiding exploring the neighborhood exhaustively with the First Improvement approach lead us to marginally increase the computing time with respect to the constructive method (10.57 versus 8.99 s on average). However, the Best Improvement approach requires almost twice the computing time (15.24 versus

[1] http://www.ics.uci.edu/mlearn/MLRepository.html

Table 1
Individual description of each instance considered in this work. For each instance, the following parameters are provided: n, total number of points; s, number of dimensions; k, number of clusters to create.

Instance Name	n	s	k
Body	507	5	2
Breast Cancer	569	30	2
Glass	214	9	7
Image Segmentation	2310	19	7
Ionosphere	351	34	2
Iris	150	4	3
Libra	360	90	15
Multiple Features Reduced	2000	240	7
Synthetic Control	600	60	6
Thyroid	215	5	3
User Knowledge	403	5	4
Vehicle	846	18	6
Vowel	871	3	3
Water	527	38	13
Wine	178	13	3
Yeast	1484	8	10
Cardiotopography	2126	24	10
MobileKSD	2855	71	56
Ozone	2536	73	21
Seismic Bumps	2584	15	19
Internet Ads	3279	1555	2
PhonesAcc	2000	3	4
PhonesGyro	2000	3	4
WatchAcc	2000	3	4
WatchGyro	2000	3	4

Table 2
Performance of the constructive method *JoinClosest* when varying the α parameter value.

Algorithm	Dev. (%)	#Best	Time (s)
JoinClosest(0.25)	1.10	1	8.99
JoinClosest(0.50)	20.51	2	9.89
JoinClosest(0.75)	22.77	0	9.85
JoinClosest(*RND*)	13.94	1	9.26

Table 3
Comparison of the two local search procedures, Best Improvement (BI) and First Improvement (FI),

Algorithm	Dev. (%)	#Best	Time (s)
JoinClosest(0.25)	31.41	0	8.99
JoinClosest(0.25) + BI	0.24	2	15.24
JoinClosest(0.25) + FI	0.07	2	10.57

8.99), without significantly improving the obtained results. Therefore, we select the First Improvement approach as the local search procedure for the final algorithm.

The next preliminary experiment is intended to evaluate the contribution of using Strategic Oscillation to further improve the best solution obtained with the GRASP procedure. As stated in Section 4, SO method requires only one parameter that controls how far from feasibility are the solutions explored during the search. In this experiment we consider the values $\beta = \{0.1, 0.25, 0.50, 0.75\}$. These values indicates the increase of each cluster size in a 10%, 25%, 50%, and 75% of the original cluster size. We do not consider larger values of β since values that exceed 100% will evaluate clusters with more than twice the original size, which is rather distant from feasibility. If so, it would be equivalent to start the search from a totally new solution.

Table 4 presents the results obtained with the aforementioned different values of β. Additionally, we have included the best GRASP variant for measuring the contribution of the Strategic Oscillation to the quality of the algorithm. As can be easily seen, every variant of the Strategic Oscillation algorithm outperforms the GRASP procedure in all metrics, barely increasing the computing time. Among SO variants, the one that increases the size of the cluster in a 75% obtains the best results. This can be mainly due to its ability to explore further regions of the search space, most of them intractable for the remaining variants. Therefore, we select $\beta = 0.75$ as the best variant of the SO algorithm.

Table 4
Results obtained by Strategic Oscillation (SO) with different increments in the cluster size.

Algorithm	Dev. (%)	#Best	Time (s)
JoinClosest(0.25) + FI	0.81	0	10.57
SO(0.10)	0.24	3	10.94
SO(0.25)	0.66	1	10.69
SO(0.50)	0.22	2	10.82
SO(0.75)	0.09	3	11.21

5.2. Comparison with the state-of-the-art procedures

The next experiment is devoted to evaluating the contribution of our best proposal by comparing it with the best previous method identified in the state of the art, which is VNS-LIMA [9]. This method is a Variable Neighborhood Search algorithm that follows the "Less Is More Approach". We additionally include a third algorithm in the comparison to verify the superiority of the proposal. In particular, we have considered an adaptation of the traditional k-means for balanced clustering [25]. This work follows the well-known k-means clustering algorithm, which is one of the most successful algorithms for clustering. Instead of selecting the closest centroid, it considers a set of clusters in which a point can be assigned, in order to satisfy the size constraint. This assignment is performed by following the Hungarian algorithm [23].

Additionally, we have executed two state-of-the-art algorithms for the classic clustering problem. Specifically, the Hierarchical Clustering with Optimal Transport (HCOT) [5] and the Continuous Greedy Randomized Adaptive Search Procedure (CGRASP) [29]. Notice that these methods do not consider the balance constraint. Therefore, once an algorithm obtains a solution, a post processing method is applied in order to make it feasible. In order to do so, each point belonging to an overloaded cluster is moved to the best cluster not yet completed.

With the aim of facilitating the comparison among algorithms, we report in Table 5 the same information than the one reported in [9]. Specifically, we show the Mean Squared Error of (MSE) of every single instance achieved with the proposed Strategic Oscillation algorithm (SO), when comparing with those obtained with HCOT, CGRASP, k-means with Hungarian algorithm (KMH), and VNS-LIMA. All the algorithms have been executed in the same computer and the same time per instance to have a fair comparison (last column of Table 5). We additionally consider in these experiments the whole set of 25 instances. Instances where a procedure is not able to produce a feasible solution are marked with an asterisk symbol ′ ∗ ′.

As we can observe, instance dimensions are rather different so it is hard to compare directly the MSE value. Therefore, we consider the average deviation with respect to the best known value since this metric is dimensionless. In particular, SO presents an average deviation of 0.38% with respect to the best known value, while the deviation of VNS-LIMA rises up to 4.84%. Finally, the HCOT, CGRASP, and KMH algorithms have a deviation of 23.70%, 54.35%, and 23.20%, respectively. Summarizing, SO is able to reach the best known solution in most of the instances (19 out of 25). Symmetrically, in the 6 instances in which SO does not obtain the best value, the corresponding result remains very close to the best known.

With the aim of studying the adaptability of the algorithms to different metrics, we evaluate the five procedures when considering an alternative metric not used during the optimization phase. In particular, the well-known Davies-Bouldin [10] index (DB) is analyzed. This index was proposed for evaluating the quality of clustering algorithms by reducing the inter-cluster similarity while increasing the intra-cluster similarity. The smaller the index value, the better the clustering. Table 6 shows the individual results of each algorithm over each instance evaluated with the Davies-Bouldin index.

These results are in line with those reported in Table 5, obtaining our method the best DB index in 16 out of 25 instances, followed by VNS-LIMA (11 out of 25). As expected, algorithms not explicitly designed for the BMSSC problem present a moderate performance. We can then conclude that the proposed Strategic Oscillation algorithm emerges as the best variant even considering this new metric.

We complement these experiments by conducting a Friedman test to determine whether there exists statistically significant differences among the compared methods or not. The resulting p-value smaller than 0.0001, in both MSE and DB metrics, indicates that the proposed algorithm is statistically better than the competitors. Table 7 reports the associated rank values for both MSE and DB for the five compared algorithms. These results confirm the superiority of the proposal when considering short computing times, which is specially relevant when considering applications where clustering is just a small part of the whole process that must be performed several times.

6. Experimental analysis

This section is devoted to deeply analyzing the parameters selected for the proposed algorithm. Specifically, we first test the robustness and reliability of the proposed algorithm by executing 30 times the SO algorithm over the 4 instances of the preliminary experimentation (Multiple Features, Vehicles, Yeast, and Image Segmentation). Considering that differences in the objective function are rather large, we use the average deviation with respect to the best value found in the 30 independent executions, since it is a dimensionless metric. We depict in Fig. 3 the associated box and whisker plot, reporting for each instance quartiles, median, minimum, and maximum values (excluding outliers). see Table 8.

Table 5
Final comparison among SO, VNS-LIMA, KMH, CGRASP and HCOT considering the MSE metric. Best values found with each method are highlighted with bold font.

Instance Name	HCOT	CGRASP	KMH	VNS	OS	Time (s)
Body	1.14E+05	1.14E+05	2.34E+05	**1.14E+05**	1.14E+05	1.08
Breast cancer	*	**1.38E+08**	1.38E+08	**1.38E+08**	**1.38E+08**	0.90
Glass	9.70E+02	9.36E+02	7.29E+02	**5.08E+02**	5.25E+02	0.27
Image Segmentation	*	2.24E+07	2.75E+07	2.14E+07	**2.14E+07**	19.24
Ionosphere	2.52E+03	2.44E+03	2.52E+03	**2.43E+03**	**2.43E+03**	0.32
Iris	**8.14E+01**	**8.14E+01**	8.90E+01	**8.14E+01**	**8.14E+01**	0.06
Libra	*	6.66E+07	6.70E+07	6.42E+07	**6.41E+07**	0.57
Multiple Features	2.04E+06	2.00E+06	2.10E+06	1.99E+06	**1.96E+06**	16.29
Synthetic Control	1.64E+06	1.09E+06	1.28E+06	1.14E+06	**1.01E+06**	0.97
Thyroid	3.82E+04	3.70E+04	3.87E+04	**3.44E+04**	**3.44E+04**	0.12
User Knowledge	8.31E+01	7.17E+01	8.06E+01	7.09E+01	**7.03E+01**	0.49
Vehicle	*	6.32E+06	4.76E+06	2.93E+06	**2.90E+06**	2.13
Vowel	*	6.47E+07	1.58E+08	7.53E+07	**6.45E+07**	2.18
Water	*	2.73E+10	**7.88E+09**	7.94E+09	7.93E+09	0.96
Wine	*	5.55E+06	3.77E+06	3.83E+06	**3.77E+06**	0.08
Yeast	6.74E+01	5.82E+01	6.01E+01	5.41E+01	**5.35E+01**	8.84
Cardiotopography	1.19E+07	2.72E+07	1.10E+07	**8.50E+06**	8.60E+06	23.24
MobileKSD	*	3.85E+10	3.28E+10	3.12E+10	**3.11E+10**	141.65
Ozone	*	3.61E+09	2.56E+09	**2.46E+09**	2.56E+09	61.93
Seismic Bumps	*	1.07E+14	2.96E+13	3.05E+13	**2.94E+13**	56.93
Internet Ads	3.76E+04	*	*	**3.74E+04**	3.75E+04	24.41
Phones Acc	4.16E+02	*	3.35E+02	3.11E+02	**2.63E+02**	20.14
Phones Gyro	3.65E+01	*	3.42E+01	3.47E+01	**3.37E+01**	13.72
Watch Acc	2.08E+03	*	1.80E+03	2.65E+03	**1.68E+03**	30.86
Watch Gyro	1.64E+01	*	1.58E+01	1.54E+01	**1.54E+01**	22.86

Table 6
Final comparison among SO, VNS-LIMA, KMH, CGRASP and HCOT considering the Davies-Bouldin metric. Best values found with each method are highlighted with bold font.

Instance Name	HCOT	CGRASP	KMH	VNS	OS	Time (s)
Body	9.25E+01	9.24E+01	8.35E+02	**9.23E+01**	9.24E+01	1.08
Breast cancer	*	**1.64E+02**	1.65E+02	**1.64E+02**	**1.64E+02**	0.90
Glass	3.02E+02	5.80E+02	9.12E+01	**3.67E+01**	6.14E+01	0.27
Image Segmentation	*	1.11E+03	9.43E+02	**8.33E+02**	1.15E+03	19.24
Ionosphere	3.07E+02	2.65E+02	3.08E+02	**2.64E+02**	**2.64E+02**	0.32
Iris	**1.64E+01**	**1.64E+01**	2.02E+01	**1.64E+01**	**1.64E+01**	0.06
Libra	*	3.24E+02	4.38E+02	**2.88E+02**	3.16E+02	0.57
Multiple Features	1.18E+03	9.83E+02	1.25E+03	9.50E+02	**8.40E+02**	16.29
Synthetic Control	1.53E+03	2.90E+02	3.67E+02	4.19E+02	**2.36E+02**	0.97
Thyroid	1.16E+02	9.86E+01	1.75E+02	**9.00E+01**	**9.00E+01**	0.12
User Knowledge	2.55E+02	1.68E+02	2.02E+02	1.56E+02	**1.42E+02**	0.49
Vehicle	*	2.64E+02	9.01E+02	1.40E+02	**1.05E+02**	2.13
Vowel	*	1.47E+02	1.08E+03	1.71E+02	**1.47E+02**	2.18
Water	*	1.58E+03	4.72E+01	2.71E+01	**2.53E+01**	0.96
Wine	*	2.72E+01	1.49E+01	1.51E+01	**1.49E+01**	0.08
Yeast	7.00E+02	5.89E+02	3.65E+02	3.65E+02	**3.63E+02**	8.84
Cardiotopography	1.41E+03	3.61E+03	7.75E+02	5.02E+02	**4.85E+02**	23.24
MobileKSD	*	1.95E+03	3.92E+02	**3.89E+02**	3.94E+02	141.65
Ozone	*	5.16E+03	2.91E+02	1.50E+02	**1.46E+02**	61.93
Seismic Bumps	*	3.58E+04	**2.11E+02**	4.23E+03	2.14E+02	56.93
Internet Ads	3.12E+04	*	*	**2.57E+04**	3.01E+04	24.41
Phones Acc	2.10E+03	*	6.98E+02	3.84E+03	**5.22E+02**	20.14
Phones Gyro	1.23E+04	*	1.81E+03	1.87E+03	**1.66E+03**	13.72
Watch Acc	1.46E+03	*	**1.19E+03**	2.30E+05	1.36E+03	30.86
Watch Gyro	1.54E+04	*	6.60E+03	**6.09E+03**	6.14E+03	22.86

Table 7
Friedman test for both MSE (Mean Squared Error) and DB (Davies-Bouldin) metrics.

Metric	HCOT	CGRASP	KMH	VNS	OS	$p < 0.01$
MSE	4.38	3.62	3.50	2.10	**1.40**	YES
DB	4.34	3.62	3.30	2.10	**1.64**	YES

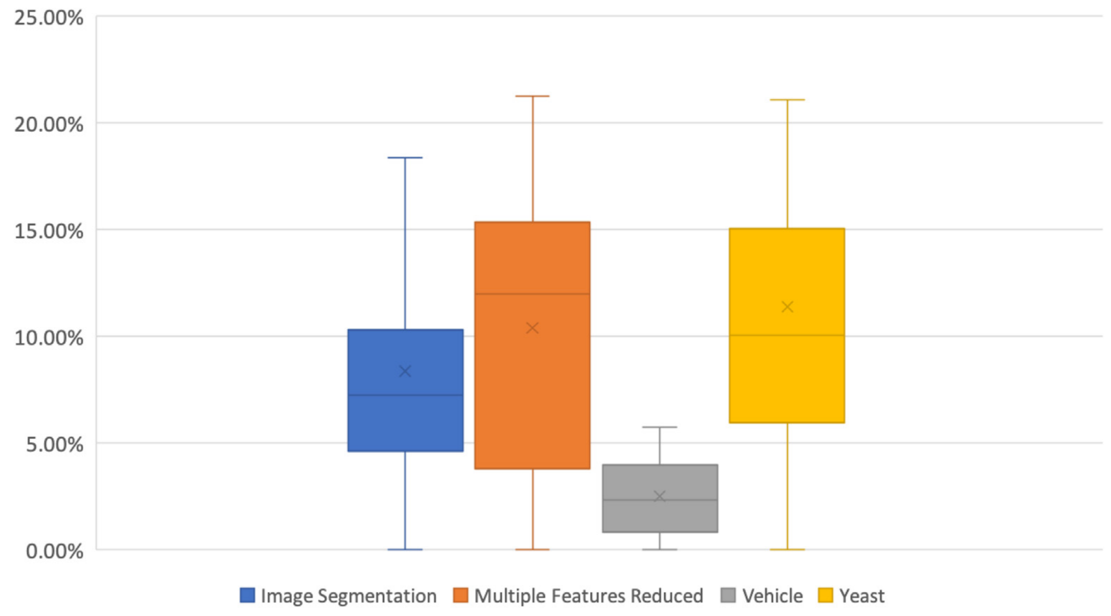


Fig. 3. Box and whisker plot for 30 independent executions.

Table 8
Sensitivity analysis for α and β parameters.

α	0.25	0.50	0.75
0.50	0.686		
0.75	0.181	0.581	
RND	0.196	0.123	0.742
β	0.10	0.25	0.50
0.25	0.012		
0.50	0.002	0.000	
0.75	0.133	0.019	0.002

As it can be derived from the results, the proposed algorithm presents a robust behavior when considering 30 independent executions in the instances considered for the preliminary experiments. As expected, the combination of GRASP with SO tries to find a balance between diversification and intensification. Specifically, in all the instances the mean (represented with an x) and median (represented with an horizontal value) values are very close, with the median under the mean in most of the cases. This result indicates that the algorithm is able to diversify the search to explore a larger portion of the search space but the intensification phase is able to lead the algorithm to high-quality solutions.

We additionally conduct a sensitivity analysis. In particular, for each parameter of the SO algorithm, namely α (the balance between greediness/randomness of the constructive method) and β (the percentage of cluster size increment inside Strategic Oscillation), we evaluate different possibilities while fixing the remaining parameters to the best values found in the preliminary experimentation. In particular, we test values $\alpha = \{RND, 0.25, 0.50, 0.75\}$ and $\beta = \{0.1, 0.25, 0.5, 0.75\}$. For each instance and each parameter setting, we execute 30 independent iterations of each algorithm.

We consider the Wilcoxon signed rank test to determine if there exists significant statistical differences among variants (in terms of the average objective function value) when we only vary a single parameter as mentioned above. The corresponding Wilcoxon test shows that the different α configurations do not significantly affect the performance of the proposal. Specifically, the associated p-values range from 0.181 to 0.742 which are considerably larger than the customary 0.05 threshold. This experiment shows that the proposed algorithm does not present a particular sensitivity with respect to this parameter.

On the other hand, observing the β parameter, there are statistical differences in performance. Therefore, this experiment justifies the election of the $\beta = 0.75$, as shown in Table 4.

To further investigate the performance of the proposed SO procedure, we conduct a convergence analysis by considering time-to-target plots (TTTPlot), which is essentially a run-time distribution [1]. The experimental hypothesis in TTTPlots is that running times fit a two parameter, or shifted, exponential distribution. Then, for a particular instance, the execution time needed to find an objective function value at least as good as a given target value is recorded. In the context of the heuristic optimization, the algorithm is determined a pre-established number of times on the selected instance and using

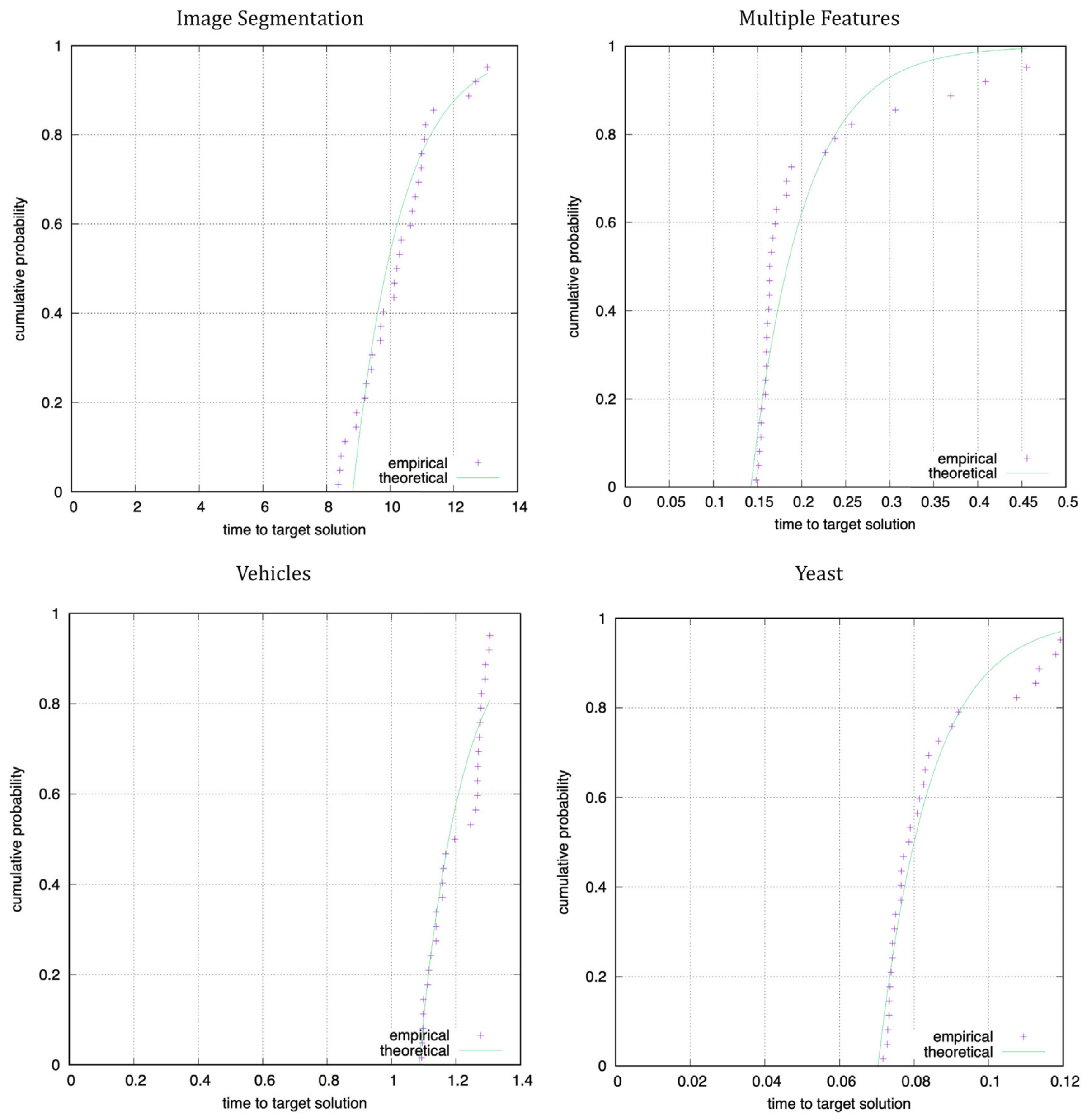


Fig. 4. Time to target plots for the preliminary instances.

the given target solution. For each of run, the random number generator is initialized with a different seed and therefore the executions are assumed to be independent. To compare the empirical and the theoretical distributions, we follow a standard graphical methodology for data analysis [7], execute our Algorithm 1 times, and recording for each instance/target pair the corresponding running time. Fig. 4 shows the TTTPlot for those instances in the set of preliminary experiments. In these figures, each value in the abscissa axis represents a running time, while each value in the ordinate axis, reports the probability of obtaining the best-known value. This experiment confirms the expected exponential run-time distribution of our SO algorithm. If we analyze the instances Multiple Features, Vehicles, and Yeast, we can observe that the probability of SO to find a solution at least at good as the target value in less than a second is close to 100%. However, regarding Image Segmentation, which is a more complex instance, this probability is near 50% when considering 10 s, requiring about 15 s to rise the probability to 100%.

7. Conclusions

We proposed a Greedy Randomized Adaptive Search Procedure (GRASP) coupled with Strategic Oscillation (SO) algorithm for the Balanced Minimum Sum-of-Squares Clustering Problem (BMSSC), which consists in grouping a set of s-dimensional points into $k$clusters maximizing the similarity among them. The experiments performed showed that SO is able to modify the search space, allowing us to explore solutions that are unattainable by using traditional heuristic procedures. This behav-

Table 9
Advantages and disadvantages of the proposed algorithm.

Advantages	Disadvantages
Best results in the literature	Not suitable if the size of the clusters is not fixed
Fast method	Does not guarantee optimality
Easily adaptable to new objective functions	Works with just one neighborhood structure
Scalable to a distributed architecture	
Independent of the dataset	

ior leads our proposal to obtain better results than the best previous method found in the state of the art. The simplicity of the method and its speed is essential when considering large amounts of data that are continuously generated (i.e., data derived from the stock exchange, social networks, etc.) and must be quickly analyzed.

The proposed algorithm presents the best results in the literature for the BMSSC problem, requiring small computing times. As we follow the GRASP methodology, the algorithm is easily scalable to a distributed system, in order to further reduce the computing times. The main limitation of our algorithm emerges when dealing with a variant in which the size of the clusters is not fixed and can vary during the execution. In that case, a deep redesign of the algorithm should be performed to adapt the method to this new problem. Furthermore, the algorithm is designed to work with a single neighborhood structure. In order to include more neighborhoods, one shall consider a more complex local search method, such as Variable Neighborhood Descent, which embeds several neighborhood structures in the same algorithm.

In order to summarize the main features of our procedure, we report in Table 9 its main advantages and disadvantages.

Focusing on the first disadvantage mentioned, this paper deals with the BMSSC, in which all the clusters share the same size. The proposed algorithm is designed to obtain high quality solutions when the cluster size is constrained. This proposal can be easily adapted to an unconstrained problem by modifying the feasibility constraint and the greedy criterion of the algorithm, in order to consider alternative moves inside the local search and constructive procedure.

As a metaheuristic algorithm, GRASP does not guarantee optimality. However, it must be born in mind that, in most real world applications, finding the optimum value is not feasible, mainly due to the complexity of the problem under consideration. Nonetheless, the proposed GRASP algorithm tries to balance solution quality and computational cost, being able to reach high quality solutions in reasonable computing time. Due to the size of the instances under consideration, using an exact solver cannot be considered.

This algorithm considers a single neighborhood structure since the results obtained are excellent, and it is not necessary to increase the computational time by including additional neighborhoods. However, the algorithm can be easily modified to consider new neighborhood structures, by extending the proposed local search procedure.

CRediT authorship contribution statement

R. Martín-Santamaría: Conceptualization, Methodology, Software. **J. Sánchez-Oro:** Conceptualization, Methodology, Software. **S. Pérez-Peló:** Conceptualization, Methodology, Software. **A. Duarte:** Conceptualization, Methodology, Software.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We would like to thank professors Costa, Aloise, and Mladenovic for kindly sending us the executable version of their algorithm and for the help configuring it. This work has been partially supported by the "Ministerio de Ciencia, Innovación y Universidades" under grant Ref. PGC2018-095322-B-C22 and "Comunidad de Madrid" and "Fondos Estructurales" of European Union with grant Ref. S2018/TCS-4566.

References

[1] R.M. Aiex, M.G. Resende, C.C. Ribeiro, Ttt plots: a perl program to create time-to-target plots, Optimiz. Lett. 1 (4) (2007) 355–366.
[2] D.J. Aloise, D. Aloise, C.T.M. Rocha, C. Ribeiro, J.R. Filho, L.S. Moura, Scheduling workover rigs for onshore oil production, Discrete Appl. Math. 154 (5) (2006) 695–702.
[3] D. Arthur, S. Vassilvitskii, k-means++: the advantages of careful seeding, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2007, pp. 1027–1035.
[4] S. Chakraborty, S. Das, Detecting meaningful clusters from high-dimensional data: a strongly consistent sparse center-based clustering approach, in: IEEE Trans. Pattern Anal. Mach. Intell., 2020.
[5] S. Chakraborty, D. Paul, S. Das, Hierarchical clustering with optimal transport, Stat. Probability Lett. 163 (2020) 108781.
[6] S. Chakraborty, D. Paul, S. Das, J. Xu, Entropy weighted power k-means clustering, in: International Conference on Artificial Intelligence and Statistics, PMLR, 2020, pp. 691–701.

[7] J.M. Chambers, Graphical methods for data analysis: 0, Chapman and Hall/CRC, 2017.
[8] E.C. Chi, K. Lange, Splitting methods for convex clustering, J. Comput. Graph. Stat. 24 (4) (2015) 994–1013.
[9] L.R. Costa, D. Aloise, N. Mladenović, Less is more: basic variable neighborhood search heuristic for balanced minimum sum-of-squares clustering, Inf. Sci. 415 (2017) 247–253.
[10] D.L. Davies, D.W. Bouldin, A cluster separation measure, IEEE Trans. Pattern Anal. Mach. Intell. 2 (1979) 224–227.
[11] J. Desrosiers, N. Mladenović, D. Villeneuve, Design of balanced mba student teams, J. Oper. Res. Soc. 56 (1) (2005) 60–66.
[12] A. Duarte, J. Sánchez-Oro, M.G.C. Resende, F. Glover, R. Martí, Greedy randomized adaptive search procedure with exterior path relinking for differential dispersion minimization, Inf. Sci. 296 (2015) 46–60.
[13] A.W.F. Edwards, L. Cavalli-Sforza, A method for cluster analysis, Biometrics 21 (2) (1965) 362–375.
[14] T.A. Feo, M.G.C. Resende, A probabilistic heuristic for a computationally difficult set covering problem, Oper. Res. Lett. 8 (2) (1989) 67–71.
[15] T.A. Feo, M.G.C. Resende, S.H. Smith, A greedy randomized adaptive search procedure for maximum independent set, Oper. Res. 42 (5) (1994) 860–878.
[16] F. Glover, J.-K. Hao, The case for strategic oscillation, Ann. Oper. Res. 183 (1) (2011) 163–173.
[17] T.F. González, On the computational complexity of clustering and related problems, in: R.F. Drenick, F. Kozin (Eds.), System Modeling and Optimization, Berlin, Heidelberg. Springer, Berlin Heidelberg, 1982, pp. 174–182..
[18] L.W. Hagen, A.B. Kahng, New spectral methods for ratio cut partitioning and clustering, IEEE Trans. CAD Integrated Circuits Syst. 11 (9) (1992) 1074–1085.
[19] A. Hinneburg, D.A. Keim, Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering, in: Proceedings of the 25th International Conference on Very Large Databases, 1999, pp. 506–517..
[20] K. Jajuga, A. Sokolowski, H.-H. Bock, Classification, clustering, and data analysis: recent advances and applications, Springer Science & Business Media, 2012.
[21] J. Jin, W. Wang, et al, Influential features pca for high dimensional clustering, Ann. Stat. 44 (6) (2016) 2323–2359.
[22] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, A.Y. Wu, An efficient k-means clustering algorithm: analysis and implementation, IEEE Trans. Pattern Anal. Mach. Intell. 24 (7) (2002) 881–892.
[23] H.W. Kuhn, The hungarian method for the assignment problem, Naval Res. Logist. Quarterly 2 (1–2) (1955) 83–97.
[24] M. Mahajan, P. Nimbhorkar, K. Varadarajan. The Planar k-Means Problem is NP-Hard, in: S. Das, R. Uehara (Eds.), WALCOM: Algorithms and Computation, Berlin, Heidelberg. Springer, Berlin Heidelberg, 2009, pp. 274–285..
[25] M.I. Malinen, P. Fränti, Balanced k-means for clustering. In: P. Fränti, G. Brown, M. Loog, F. Escolano, M. Pelillo (Eds.), Structural, Syntactic, and Statistical Pattern Recognition, Berlin, Heidelberg. Springer, Berlin Heidelberg, 2014, pp. 32–41..
[26] R. Martí, A. Martínez-Gavara, J. Sánchez-Oro, A. Duarte, Tabu search for the dynamic Bipartite Drawing Problem, Comput. OR 91 (2018) 1–12.
[27] S.M. Mohammed, K. Jacksi, S. Zeebaree, A state-of-the-art survey on semantic similarity for document clustering using glove and density-based algorithms, Indonesian J. Electrical Eng. Comput. Sci. 22 (1) (2021) 552–562.
[28] A. Pyatkin, D. Aloise, N. Mladenović, NP-Hardness of balanced minimum sum-of-squares clustering, Pattern Recogn. Lett. 97 (2017) 44–45.
[29] E. Queiroga, A. Subramanian, F. dos Anjos, L. Cabral, Continuous greedy randomized adaptive search procedure for data clustering, Appl. Soft Comput. 72 (2018) 43–55.
[30] M. Sevaux, A. Rossi, M. Soto, A. Duarte, R. Martí, GRASP with ejection chains for the dynamic memory allocation in embedded systems, Soft Comput. 18 (8) (2014) 1515–1527.
[31] L. Shapiro, G. Stockman, Computer Vision, Prentice-Hall, Upper Saddle River, NJ, 2001.
[32] H. Steinhaus, Sur la division des corps matériels en parties, Bull. Acad. Polon. Sci. Cl. III. 4 (1956) 801–804.
[33] W. Su, J. Hu, C. Lin, S.X. Shen, SLA-Aware Tenant Placement and Dynamic Resource Provision in SaaS, in: J.A. Miller, H. Zhu (Eds.), ICWS, IEEE Computer Society, 2015, pp. 615–622.
[34] K. Wong, A Short Survey on Data Clustering Algorithms, in: 2015 Second International Conference on Soft Computing and Machine Intelligence (ISCMI) , 2015, pp. 64–68.
[35] K. Wong, Z. Zhang, SNPdryad: predicting deleterious non-synonymous human SNPs using only orthologous protein sequences, Bioinformatics 30 (8) (2014) 1112–1119.
[36] A.E. Xavier, V.L. Xavier, Solving the minimum sum-of-squares clustering problem by hyperbolic smoothing and partition into boundary and gravitational regions, Pattern Recogn. 44 (1) (2011) 70–77.
[37] H. Xiong, J. Wu, J. Chen, K-means clustering versus validation measures: a data-distribution perspective, IEEE Trans. Systems, Man, and Cybernetics, Part B 39 (2) (2009) 318–331.
[38] J. Xu, K. Lange, Power k-means clustering, in: International Conference on Machine Learning, PMLR, 2019, pp. 6921–6931.
[39] R. Xu, D. Wunsch, Survey of clustering algorithms, IEEE Trans. Neural Networks 16 (3) (2005) 645–678.

Chapter 7

An Efficient Algorithm for Crowd Logistics Optimization

Title	An Efficient Algorithm for Crowd Logistics Optimization
Authors	Raúl Martín-Santamaría, Ana Dolores López Sánchez, María Luisa Delgado Jalón and José Manuel Colmenar Verdugo
Publication date	2021
Journal	Mathematics
Publisher	MDPI
ISBN/ISSN	2227-7390
Impact Factor	2.592 (2021)
Rank by Impact Factor	21/333 (Q1, Mathematics)
DOI	`https://doi.org/10.3390/math9050509`

mathematics

MDPI

Article

An Efficient Algorithm for Crowd Logistics Optimization

Raúl Martín-Santamaría [1], Ana D. López-Sánchez [2], María Luisa Delgado-Jalón [1,3] and J. Manuel Colmenar [1,*]

1 Computer Science and Statistics Department, Universidad Rey Juan Carlos, Tulipán s/n, Móstoles, 28933 Madrid, Spain; raul.martin@urjc.es (R.M.-S.); marialuisa.delgado@urjc.es (M.L.D.-J.)
2 Department of Economics, Quantitative Methods and Economic History, Universidad Pablo de Olavide, Ctra. de Utrera km 1, 41013 Sevilla, Spain; adlopsan@upo.es
3 Department of Economics, Quantitative Methods and Economic History, Universidad Rey Juan Carlos, Tulipán s/n, Móstoles, 28933 Madrid, Spain
* Correspondence: josemanuel.colmenar@urjc.es

Abstract: Crowd logistics is a recent trend that proposes the participation of ordinary people in the distribution process of products and goods. This idea is becoming increasingly important to both delivery and retail companies, because it allows them to reduce their delivery costs and, hence, to increase the sustainability of the company. One way to obtain these reductions is to hire external drivers who use their own vehicles to make deliveries to destinations which are close to their daily trips from work to home, for instance. This situation is modelled as the Vehicle Routing Problem with Occasional Drivers (VRPOD), which seeks to minimize the total cost incurred to perform the deliveries using vehicles belonging to the company and occasionally hiring regular citizens to make just one delivery. However, the integration of this features into the distribution system of a company requires a fast and efficient algorithm. In this paper, we propose three different implementations based on the Iterated Local Search algorithm that are able to outperform the state-of-art of this problem with regard to the quality performance. Besides, our proposal is a light-weight algorithm which can produce results in small computation times, allowing its integration into corporate information systems.

Keywords: vehicle routing problem; crowd logistics; crowdshipping; occasional drivers; iterated local search

check for updates

Citation: Martín-Santamaría, R.; López-Sánchez, A.D.; Delgado-Jalón, M.L.; Colmenar, J.M. An Efficient Algorithm for Crowd Logistics Optimization. *Mathematics* **2021**, *9*, 509. https://doi.org/10.3390/math9050509

Academic Editor: Fabio Caraffini

Received: 27 January 2021
Accepted: 23 February 2021
Published: 2 March 2021

1. Introduction

Nowadays, many people use e-commerce to buy and sell all kind of goods, products and services. The 24/7 availability of the websites, the wide array of products and services, the easy reachability to get any of them in any place, the easy way of comparing prices and the possibility of gather opinions from other customers are some of the advantages that support the use of the e-commerce. These reasons, together with the lack of time of most citizens, make e-commerce continue to grow, which leads to an increase on the delivery services, especially in the last-mile operations [1].

Both delivery companies and retailers which also distribute their products strive to minimize the total cost and the delivery time to be more efficient [2]. However, the increase of delivery operations impacts on the sustainability of a company, either by enlarging the routes or by performing door-to-door distributions, which could require increasing the number of vehicles attending customers [3]. In an early stage of the last-mile development, ref. [4] held that the cost related to the last-mile operation may range between 13 and 75% of the total distribution cost. Therefore, the optimization of this step of the supply chain led to an important reduction on the distribution costs for many delivery companies that focused on solving the last-mile problem to reduce costs. Since then, many delivery companies have focused on reducing this cost, but it is important to emphasize that this range may vary significantly depending on the specific problem under consideration.

Recently, a new trend called crowd logistics is gaining relevance [5–7]. The idea behind this concept is to favor the participation of ordinary citizens in the distribution of goods. The authors of [6] distinguish between four types of crowd logistics: crowd storage, crowd local delivery, crowd freight shipping and crowd freight forwarding. We will only focus on one of them: the crowd local delivery or crowdshipping, as called by other authors [8]. In particular, the key concept of crowdshipping here is to either deliver orders through other customers, or hire regular people close to the delivery route to occasionally perform a delivery on behalf of a logistics operator. The implementation of this idea may contribute to the sustainability of a company because it will reduce the logistics network [6] and, eventually, reduce the urban traffic levels [9] as well as the logistics costs [10]. In addition, this can be an opportunity to obtain extra incomes for the occasional couriers at the cost of slightly modifying their daily route from work to home or vice versa [6]. Note that crowshipping can be seen as an example of sustainable transportation apart from other typical examples such as walking, cycling, carpooling, car sharing, or green vehicles [11].

The concept of crowd logistics is mainly related to the vehicle routing problem (VRP). The VRP aims to find the best routes to satisfy the demand of a set of customers, given a fleet of vehicles [12,13]. Therefore, since crowd logistics involves the last-mile operations, it can be tackled from the VRP point of view.

In this paper we propose a light-weight and efficient algorithm to optimize the last-mile logistics including the concept of crowdshipping. As it is well-known, the last-mile delivery problem consists on the transportation of the goods from the warehouse, called depot in this work, and the final destination, usually the customer's home or business. Last-mile problems are considered very important regarding sustainability since they involve the less efficient phase of the logistic process [14]. Furthermore, in this work, we consider those deliveries to be performed by ordinary citizens, the occasional drivers, as part of a crowdshipping strategy, in addition to the own staff of the delivery company. Previous works like [15] describe the impact on sustainability of this kind of distribution models.

Given the small computation time and low complexity of our algorithm, it can be included in a corporate information system with the objective of optimizing a set of delivery orders taking into account both delivery routes and occasional couriers. Hence, a company will be able to reduce its costs and increase its sustainability levels by hiring occasional couriers, since the proposed algorithm is able to optimize the last mile routes taking into account the collaboration of occasional drivers.

In order to assess our proposal, we have studied the Vehicle Routing Problem with Occasional Drivers (VRPOD). VRPOD is able to model last-mile situations appearing in delivery companies that allow crowdshipping in addition to their own staff. The VRPOD assumes that the company has a fleet of vehicles handled by regular drivers who make deliveries limited by the capacity of the vehicles. Furthermore, the company is able to hire a number of occasional drivers to make a single delivery using their own vehicles. The objective of the VRPOD is to minimize the total cost, calculated as the sum of the costs incurred by the regular drivers performing traditional routes, beginning and ending at the depot, plus the cost of paying the occasional drivers, since they provide their service in exchange of remuneration. To the best of our knowledge, this problem was firstly defined by [16], who studied the potential benefits of including occasional drivers to make deliveries as a way of crowdshipping. The authors considered two compensation schemes to pay fixed fees to every occasional driver.

Since then, new variants of the VRPOD have been analyzed. In the work proposed in [17], the authors considered that occasional drivers appear dynamically and they assume that stochastic information is known about this behaviour. Furthermore, occasional drivers could serve one or more of the customers. The authors propose a stochastic mixed-integer programming formulation to solve the problem. They study the effects of uncertainty to design the routes when the occasional drivers can appear later in the day. A similar work by [18] includes two aspects: the possibility for occasional drivers to make multiple deliveries and the time windows for the customers and so the occasional drivers. The authors study the advantages of employing two different alternatives: occasional drivers that are allowed to perform multiple deliveries and occasional drivers that can split the deliveries. Their proposal is proven using two different mathematical models. Later on, a variant of the VRPOD in which occasional drivers may accept or reject the assigned delivery with a certain probability was presented in [19]. The authors solve the problem with a bi-level methodology in which they start by including all the deliveries in regular routes without the use of occasional drivers, and then include deliveries to occasional drivers taking into account their acceptance probabilities modelled using a uniform distribution.

There exist other problems related to the VRPOD although they present some differences since they focus on the crowdshipping. In [20], a variant of the dynamic pickup and delivery problem is introduced, in which occasional drivers dynamically appear to make deliveries in exchange for a small compensation. They study how profitable is the use of a platform that matches deliveries and occasional drivers in order to facilitate on-demand delivery. Furthermore, they use regular routes to serve customers for which the use of an occasional driver is not feasible or not efficient. They solve the problem using a rolling horizon framework approach to determine the matches based on the available information, and propose an exact solution approach to solve the matching problem each time new information appears. Other similar problem dealing with the crowdshipping is studied by [21]. In this paper the authors do not consider the use of regular routes to perform the deliveries but they just assume the use of occasional drivers (or *crowdshippers*), who can accept more than one delivery to transport more than one item meanwhile the vehicle capacity is not exceeded. They propose an exact solution methodology to solve the specific problem.

Among all the previous works and approaches to the VRPOD, we have selected the definition stated in [16] in order to assess our proposal. As it will be shown, we propose an algorithm able to either obtain optimal solutions when the optimal value is known, or to improve the best-known solutions, providing high-quality results in a reasonable amount of time for the VRPOD. Hence, our main contribution after [16] is the new algorithmic design, which is fast enough to be included in corporate information systems, and obtains better solutions than the previous work. To this end, we propose three different variants of the Iterated Local Search (ILS) algorithm, since this methodology has been successfully used to deal with many different variants of vehicle routing problems (VRP). For instance, in [22] an ILS algorithm solves the VRP with backhauls, being able to obtain high-quality solutions in short computational time. In [23], an ILS method is proposed to address another variant of this type of problems, the Multi-Commodity Multi-Trip VRP with Time Windows, outperforming the previous algorithm. Finally, in [24], the proposed ILS deals with the Split Delivery VRP obtaining highly competitive results.

Specifically, in this paper we present a multi-start ILS algorithm where a greedy randomized constructive method is proposed, and five different neighborhoods are combined to form a new extended neighborhood, which is explored by the local search step of ILS for the solution of the VRPOD. Besides, three perturbation strategies have been proposed and analyzed. In addition to the customary ILS implementation, we propose a straightforward parallelization of the ILS method, and a collaboration scheme where different ILS configurations cooperate in parallel. All these contributions have been assessed in a set of preliminary experiments where the final configuration and the parameter values for the algorithm have been determined. Finally, a detailed comparison with the state of the art is performed.

The rest of the paper is organized as follows. Section 2 describes the VRPOD problem. Section 3 details the algorithmic proposal implemented to solve the problem under study. Section 4 provides an extensive computational study, and performs a comparison against the state of the art. Finally, Section 5 draws the conclusions of this work and discusses future research.

2. Problem Definition

The VRPOD can be formally stated as follows. Let $G = (V, A)$ be a complete directed graph, where $V = \{0, K, C\}$ is the set of vertices, with vertex 0 as the depot, $K = \{1, \ldots, k\}$ the set of vertices representing the location of the occasional drivers and $C = \{1, \ldots, n\}$ the set of vertices corresponding to the location of customers ($|V| = 1 + k + n$). Each node $i \in C$ has an associated positive demand $q_i > 0$. Furthermore, $A = \{(i, j) : i, j \in V, i \neq j\}$ is the arc set, where (i, j) represents a path between vertices i and j. For each pair $(i, j) \in A$, let $d_{ij} \geq 0$ be the length of the shortest path that connects i and j. The cost of a route is the sum of the distances between consecutive nodes, including the depot.

Customers can be served by regular drivers on routes starting and ending at the depot. We consider their vehicles to have a limited capacity Q. This variant of the problem allows to hire occasional drivers to make a single delivery to a customer if the following condition is satisfied. An occasional driver $k \in K$ can serve customer $i \in C$ if $d_{0i} + d_{ik} \leq \zeta d_{0k}$ with $\zeta \geq 1$. In other words, if the extra distance to get the occasional driver from the depot through the customer i is less than or equal to $(\zeta - 1)$ times the direct distance from the depot to the occasional destination's location; $d_{0i} + d_{ik} - d_{0k} \leq (\zeta - 1)d_{0k}$. Therefore, ζ is referred as the flexibility of the occasional drivers. It is important to emphasize that a trip of an occasional driver is measured as the distance traveled from the depot to the customer and from the customer to the occasional driver location. Furthermore, it is assumed that the capacity of any occasional driver is enough to satisfy the demand of any customer but one occasional driver can serve a maximum of one customer.

The objective of the VRPOD is to minimize the aggregated cost incurred by regular and occasional drivers. Notice that an occasional driver is paid only if he/she serves a customer. This payment to the occasional driver is computed considering two different schemes, namely Scheme I and Scheme II. Both take into account a compensation rate denoted by ρ. In Scheme I, the compensation does not depend on the occasional drivers' destination. Thus, every occasional driver receives ρd_{0i} as compensation for making a delivery to customer i. In this scheme, the compensation rate is limited to $0 < \rho < 1$. Therefore, this scheme only requires to know the location of the customers, which means that occasional drivers serving customers far from their locations are not compensated for the extra mileage incurred. As an alternative, Scheme II defines a compensation that actually depends on the destination of the occasional drivers, the customer location and the depot. In this case, each occasional driver k receives a compensation of $\rho(d_{0i} + d_{ik} - d_{0k})$ for the extra mileage incurred for serving the customer i, with $\rho \geq 1$. This variant is more difficult to put into practice since the company needs to know the destination of the occasional drivers. For further details, see [16] where a mathematical formulation is included.

In [16], the previously explained compensation schemes were studied to assess the advantages and disadvantages of implementing both schemes, taking also into account the economical benefits for the companies depending on the number and flexibility of the occasional drivers. A detailed formulation of this problem can be found in [16]. Despite that realistic situations may generate different compensation schemes depending on each specific delivery company payment policy, we will assess the efficiency of our proposed algorithms by means of a comparison with [16]. Consequently, we consider that the occasional drivers can only visit one customer since they are not professional couriers because splitting the deliveries would be more expensive for the delivery company. Besides, if the occasional driver is available to perform a delivery, then the probability of rejecting this service is uncertain and, likely, very low. Hence, we do not take into account this feature.

3. Algorithmic Proposal

In this paper, an Iterated Local Search (ILS) algorithm is proposed to tackle the VRPOD problem. This metaheuristic, see [25], proposes the coupling of a local search method with a perturbation or disturbance process that allows the local search to escape from local optima. We selected this algorithm due to its simple design, and, at the same time, very effective performance. In fact, its design favors the implementation of parallel cooperative schemes, as will be later explained. In particular, we have used a multi-start approach for the ILS which accepts four different parameters: nc, which determines the number of constructions to be generated, that is, the number of starts of the algorithm; α, which controls the greediness of the construction of solutions; np, which corresponds to the number of perturbations that will be performed; and β, which is the perturbation intensity.

The pseudo-code of our proposal is shown in Algorithm 1. As stated before, ILS iterates nc times generating a new solution by means of the constructive method (step 3) on each iteration. Then, a new loop begins, which will disturb and improve the solution np times (steps 4 to 10). After the perturbation and improvement (steps 5 and 6), the objective function value of the resulting solution S'' is compared with the current solution previous to the perturbation, S. If the new solution is better, the current best solution is updated (steps 7 to 9). Finally, the best solution is returned in step 15.

Algorithm 1 ILS(nc, α, np, β)

```
1:  $S^\star \leftarrow \emptyset$
2:  for 1...$nc$ do
3:      $S \leftarrow ConstructiveMethod(\alpha)$
4:      for 1...$np$ do
5:          $S' \leftarrow Perturbation(S, \beta)$
6:          $S'' \leftarrow LocalSearch(S')$
7:          if $f(S'') < f(S)$ then
8:              $S \leftarrow S''$
9:          end if
10:     end for
11:     if $f(S) < f(S^\star)$ then
12:         $S^\star \leftarrow S$
13:     end if
14: end for
15: return $S^\star$
```

Next, each one of the components of the ILS method will be described, as well as their complexity both in terms of time and space. Notice that the complexity of ILS is the maximum of its components.

3.1. Constructive Method

In order to generate a variety of different and good-quality initial solutions, a GRASP methodology has been implemented. GRASP (Greedy Randomized Adaptive Search Procedure) was proposed in [26] and formally defined in [27] as an iterative algorithm with two phases: a randomized construction phase that uses a greedy function to build solutions followed by a local search phase. Two main reasons lead us to select the GRASP methodology for the constructive phase: on the one hand, it is able to produce high-quality and diverse solutions by tunning the value of the α parameter, making possible to explore wider regions of the solutions space; on the other hand, its simple design makes it fast, being able to obtain a large number of initial feasible solutions in tiny computing times.

Given that the ILS procedure performs its own local search after the perturbation, we only execute the randomized construction phase in the constructive method.

A solution for the VRPOD is represented as a set S of assignments corresponding either to routes of regular vehicles or to occasional drivers, considering that each occasional driver can attend only one customer, and each customer is attended only once. Hence, we propose a greedy function $g(S, a_c)$ for the GRASP construction phase. This function calculates the increase of the objective cost value in a given solution S due to a route assignment a_c, being c a customer of the instance. In this context, a_c represents any valid assignment that does not break any problem constraint: each customer can be assigned either to any existing route (in any position, as long as the maximum capacity is not exceeded), to a new route or to any occasional driver available for the given customer.

Algorithm 2 details the pseudo-code of the proposed constructive method, which adds assignments to an initially empty solution S (step 1). The candidate list CL is created by including all possible assignments for each customer. We represent this process of obtaining all the assignments for the set of customers C with the method *ObtainValidAssignments* shown in step 2. The constructive procedure iterates until the CL is empty, that is, all customers are assigned either to a regular route or to an occasional driver (steps 3–11). At each iteration of the construction, all the assignments in CL are evaluated with the greedy function, $g(S, a_c)$, obtaining the best and worst values, g_{min} and g_{max}, respectively (steps 4 and 5) to calculate the threshold, th (step 6). This threshold determines which assignments enter to the restricted candidate list, RCL (step 7). The method is able to control the balance between greediness and randomness by means of the parameter α, with $0 \leq \alpha \leq 1$. If $\alpha = 0$, then only those assignments with the best value (g_{min}) are included in RCL, which is the full greedy case. If $\alpha = 1$ then the RCL will contain all the candidates and, therefore, the method will be completely random. Once the RCL is filled with assignments, one of them is randomly selected following an uniform distribution (step 8), whose corresponding customer is denoted as c'. This assignment is then added to the current solution in step 9, and the CL is updated by removing all the assignments of the selected customer (step 10). This process is repeated until there are no valid assignments in the CL, which only happens after every customer has been assigned either to a regular route or an occasional driver. Therefore, the space complexity of the GRASP constructive method is $O(|V|)$, as the data structures size scales linearly with the number of customers and occasional drivers, while the time complexity is $O(|C| \times |V|)$.

Algorithm 2 ConstructiveMethod(α)

```
1:  $S \leftarrow \emptyset$
2:  $CL \leftarrow ObtainValidAssignments(C)$
3:  while $CL \neq \emptyset$ do
4:      $g_{min} = \min_{v \in CL} g(S, a_c)$
5:      $g_{max} = \max_{v \in CL} g(S, a_c)$
6:      $th \leftarrow g_{min} + \alpha(g_{max} - g_{min})$
7:      $RCL \leftarrow \{a_c : v \in CL \wedge g(S, a_c) \leq th\}$
8:      $a_{c'} \leftarrow SelectRandom(RCL)$
9:      $S \leftarrow S \cup \{a_{c'}\}$
10:     $CL \leftarrow CL \setminus \{a_c \in CL : c = c'\}$
11: end while
12: return $S$
```

3.2. Local Search

Once the construction of a solution is detailed, the local search procedure (step 6 of Algorithm 1) is next defined. In general terms, a local search algorithm traverses a neighborhood of solutions returning the best one, which is known as the local optimum. A neighborhood of solutions consists of the set of solutions that can be reached after applying a move to the current solution. To take advantage of the problem knowledge, our algorithm considers five different neighborhoods, $\mathcal{N}_1$ to $\mathcal{N}_5$. Therefore, the exploration of several different neighbourhood structures is preferred instead of just one, in order to reach high-quality solutions. The different neighborhoods are defined by the following moves, where all of them but **2-opt** were also used in [16]:

2-opt: a sub-sequence of a route is reversed [28]. Figure 1 shows a simple example where the subtour delimited by customers B and E is reversed. This move produces the neighborhood $\mathcal{N}_1$. The space and time complexity of completely exploring this neighborhood, are $O(1)$ and $O(|C|^2)$, respectively.

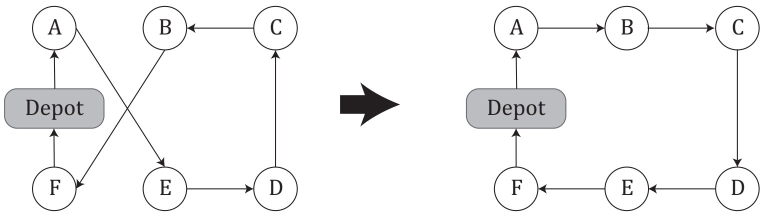


Figure 1. Example of 2-opt move between B and E nodes, which implies that edges $\overline{AE}$ and $\overline{BF}$ are removed and edges $\overline{AB}$ and $\overline{EF}$ are inserted.

1-move: a customer served by a regular route is inserted into a different regular route. Figure 2 shows how customer B is included in a different route. This move produces the neighborhood $\mathcal{N}_2$. The space and time complexity of completely exploring this neighborhood, are $O(1)$ and $O(|C|^2)$, respectively.

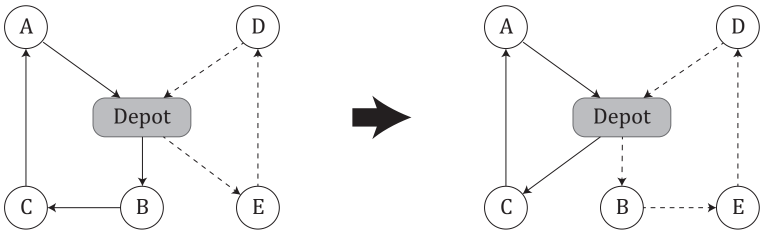


Figure 2. Example of 1-move of B from solid line route to dashed line route.

Swap-move: a pair of customers served by different regular routes are exchanged. Figure 3 shows the swap of customers B and E. This move produces the neighborhood $\mathcal{N}_3$. The space and time complexity of completely exploring this neighborhood, are $O(1)$ and $O(|C|^2)$, respectively.

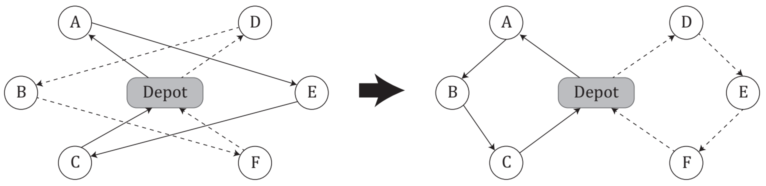


Figure 3. Example of swap move between B and E nodes.

In-move: a customer served by an occasional driver is included in a regular route. As can be seen in Figure 4 the customer B initially visited by an occasional driver will be served in a regular route after the in-move. This move produces the neighborhood $\mathcal{N}_4$.

The space and time complexity of completely exploring this neighborhood, are $O(1)$ and $O(|K| \times |C|)$, respectively.

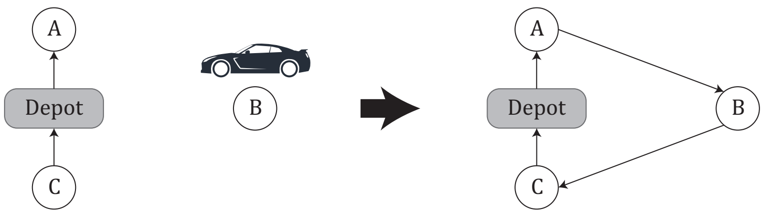


Figure 4. Example of In-move of B.

Out-move: a customer served by a regular route is assigned to an occasional driver. Figure 5 shows how customer B, initially visited by a regular route, is now served by an occasional driver. This move produces the neighborhood $\mathcal{N}_5$. The space and time complexity of completely exploring this neighborhood, are $O(1)$ and $O(|K| \times |C|)$, respectively.

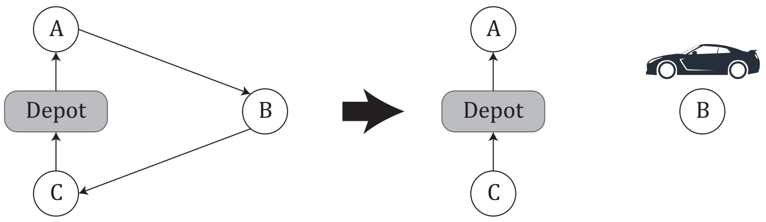


Figure 5. Example of Out-move of B.

The proposed local search method considers an extended neighborhood formed by the five defined neighborhoods. Algorithm 3 presents the pseudo-code of our proposal. As seen in the algorithm, the method iterates while the current solution is improved (steps 3 to 10). Hence, given an incumbent solution S, the five neighborhoods previously defined are explored in step 4 obtaining S', which is the best solution of the extended neighborhood. Then, it is compared with the best solution $S^\star$ in step 5, updating $S^\star$ if necessary in the following step. If no improvement was made, the guard variable is changed in step 8. At the end, the algorithm returns the local optimum $S^\star$ in step 11.

Algorithm 3 LocalSearch (S)

```
1:  $S^\star \leftarrow \emptyset$
2:  improve ← true
3:  **while** improve **do**
4:      $S' \leftarrow \arg\min_{S \in \bigcup_{i=1}^{5} \mathcal{N}_i(S)} f(S)$
5:      **if** $f(S') < f(S^\star)$ **then**
6:          $S^\star \leftarrow S'$
7:      **else**
8:          improve ← false
9:      **end if**
10: **end while**
11: **return** $S^\star$
```

3.3. Perturbation Procedures

Another important step of the ILS algorithm is the way in which a solution is perturbed or modified. Given that this problem involves routes and occasional drivers assignations, several different perturbations can be explored. Among them, three different perturbation procedures are proposed in this paper, motivated by the need to reach a solution different from the incumbent one and different to its neighbors, considering the neighborhoods previously defined. The proposed perturbation procedures are next described:

RandomMove. A move from the five previously defined neighborhoods is randomly selected and executed, without evaluating the performance impact over the objective function. This perturbation is applied a fixed number of times, defined by the β parameter. The complexity of this perturbation method corresponds with the time complexity of the used neighborhood.

RouteCost. This strategy firstly ranks the routes by their cost per customer, and then selects a route according to a probability distribution. The probability p_{r_i} of choosing a certain route r_i is given by Equation (1). This approach to select the route to remove is analogous to the one followed in [29] and [30] for the construction phase. In our implementation, removing a route has a time complexity of $O(|C|)$.

$$p_{r_i} = \frac{z_{r_i}}{\sum\limits_{r_j \in R} z_{r_j}} \tag{1}$$

where z_{r_i} represents the cost per customer of route r_i, as seen in Equation (2), in which $f(r)$ represents the cost of a given route r, and $|C_r|$ the number of customers attended by a route r. In short, as a proportion, the more costly a route is per customer, the more likely it will be destroyed.

$$z_r = \frac{f(r)}{|C_r|} \tag{2}$$

In case that all the customers where removed from a route, the route is deleted. This process is repeated β times, producing a number of unassigned customers. Then, those customers are reassigned using the proposed constructive method (see Section 3.1).

RandomDeassign. It randomly selects β customers following an uniform distribution, and their assignments are removed from the solution. Then, these customers are reassigned using the constructive method used in the ILS algorithm. In our implementation, removing a random set of customers from a given solution has a time complexity of $O(|C|^2)$.

All the three perturbation methods require an input parameter which, for the sake of clarity, we have labeled as β. This parameter determines the perturbation size, which has different meaning on each perturbation method, as explained above. Hence, the β values analyzed in the experimental experience will be selected accordingly.

3.4. Parallel Cooperation Proposal

The parallel implementation of an algorithm is usually a straightforward task that allows the researcher to make use of the full performance of the computer where the algorithm is run. Moreover, as shown by many works in the literature, parallelism can contribute to the optimization search. For instance, in [31], a parallel Variable Neighborhood Search (VNS) approach is presented, where a cooperation among threads is developed on a master-slave scheme. The authors applied this proposal to successfully solve a well-studied location problem, the p-median problem. Several cooperative schemes for VNS are also studied in [32], showing that cooperation reaches better results than the straightforward parallelization in the obnoxious p-median problem.

Based on the previously exposed ideas, we propose a cooperation scheme for the parallel implementation of the ILS method. This cooperation is shown in Figure 6. As it can be seen, the multi-start ILS execution is divided into N *workers*, namely ILS_1 to ILS_N. Each worker will execute independently on a different thread following the implementation shown in Algorithm 1, but applying the cooperation scheme. In particular, each worker creates a solution with the constructive procedure and, then, executes the internal *for* loop, which corresponds to steps 4 to 10 in the algorithm, labelled as $ILSLoop$ in the figure. After a given number of executions of the loop, a migration of solutions is performed. In this cooperation, each worker ILS_i with $i = 1...N$ "pushes" (sends) its current best solution to a FIFO queue, q_i, from which the following worker will "pull" (receive) a solution. Notice that ILS_N sends its solution to ILS_1, creating a ring topology. Once a solution is taken from the queue, the ILS loop executes on the incoming solution until the following migration or the execution ends.

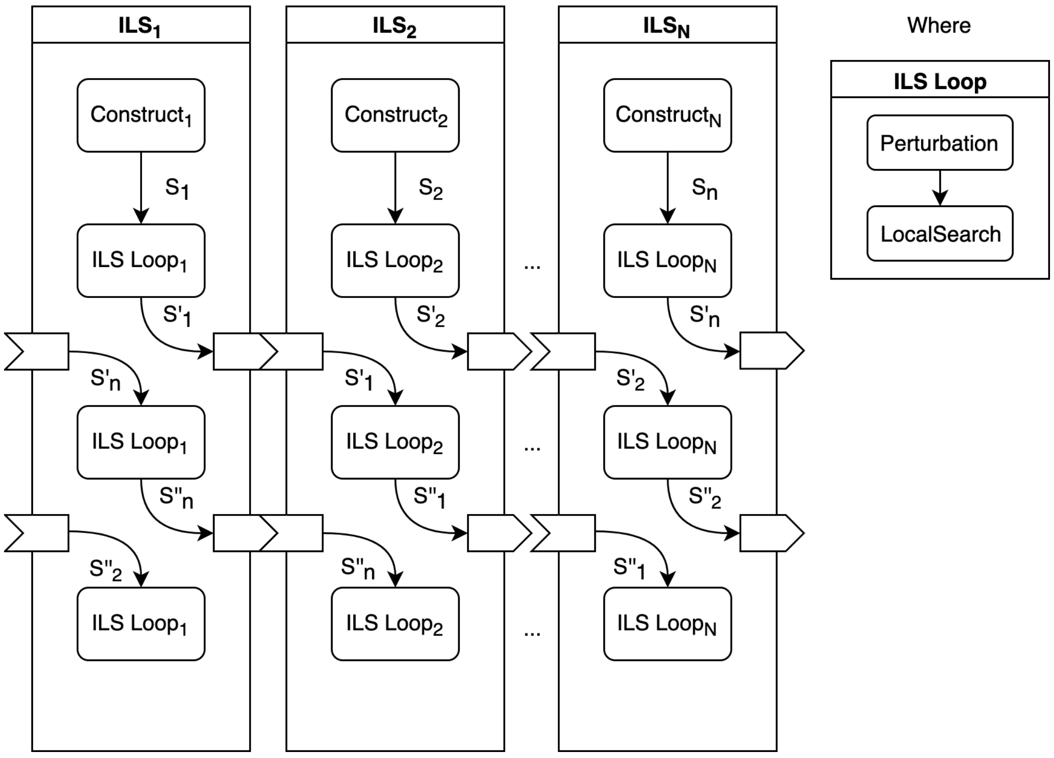


Figure 6. Parallel cooperation scheme for the ILS based on solution migrations.

The decoupling of workers by means of the queues makes this scheme very flexible, allowing different cooperative structures like master-slave or full connection [32]. However, we propose the ring configuration and the concept of *round*. A round is completed when a solution has visited every worker once. Therefore, if we set the number of rounds to two, each solution will visit each worker twice. In order to honor the total number of allowed perturbations, the algorithm will divide the number of iterations of the original *for* loop, given by the *np* parameter (see step 4 in Algorithm 1) by $(N \cdot rounds)$.

As it will be shown in the next section, the main advantage of this proposal is being able to apply different configurations on each worker. Besides, given that the queues take care of the synchronization of the threads, the execution time will be determined by the slowest worker.

4. Computational Results

This section presents and discusses the computational experience conducted with the algorithms proposed in this paper. Firstly, we describe a set of preliminary experiments that allows us to tune the parameters of the algorithm. Then, we compare the performance of our proposal against the state of the art, which was stated in [16].

In order to perform a fair comparison, we have used the very same set of instances as the previous authors. In particular, they consider six types of instances: types C101 and C201, where customers are clustered; types R101 and R201, where customers are randomly distributed; and types RC101 and RC201, where customers are partially clustered and partially randomly distributed. Following the approach from [16], we generated the different instances among the mentioned six types using the corresponding values for the parameters that characterize each instance. These parameters and their values are the following: the number of occasional drivers, $|K|$, with $|K| = \{13, 25, 50, 100\}$); the compensation rate, ρ, with $\rho = \{0.05, 0.10, 0.20\}$ and $\rho = \{1.2, 1.4, 1.6\}$ for the compensation schemes I and II, respectively; and the flexibility of the occasional drivers, ζ, with $\zeta = \{1.1, 1.2, 1.3, 1.4, 1.5, 1.6\}$. The combination of the values of the parameters across the six types of instances according to [16] produced a total number of 480 instances.

The experiments were run on a machine provided with a Ryzen 7 1700 CPU running at 3 GHz, with 16GB RAM. All the algorithms are implemented in Java 11.

4.1. Preliminary Experimentation

In order to select the best combination of parameters for our proposed algorithms, a representative subset of 70 instances out of a total number of 480 instances, was selected having the following final distribution: 8 instances with $|K| = 13$, 9 instances with $|K| = 25$, 26 instances with $|K| = 50$; and 27 instances with $|K| = 100$. The selection was made by randomly picking an instance for each combination of $|K|$, ζ and ρ.

The first preliminary experiment is devoted to tuning the generation of the initial solution. As stated in Section 3.1, a GRASP approach is proposed. Therefore, it is required to determine the best value of the α parameter for this method. Tables 1 and 2 show the results obtained when solutions are built using the GRASP constructive method and the constructive method coupled with the proposed local search, respectively. In particular, 10,000 constructions are generated in both experiments. The first column of both tables contains the different values of the parameter α that have been studied: $\alpha = \{0, 0.25, 0.5, 0.75, 1, Random\}$, where *Random* means that a value for α was randomly selected at each iteration, following a uniform distribution. Besides, the number of times that the algorithm is able to attain the best value is shown in the second column (#B.), being the best value the minimum value found by any of the compared algorithms in each experiment; the third column averages the best costs obtained across the 70 instances (Cost); and, finally, the last column shows the average computation time in seconds (T(s)).

Table 1. Performance of the different values of α for the proposed constructive method after 10,000 iterations.

α	#B.	Cost	T(s)
0.00	57	542.3	5.47
0.25	0	852.3	7.20
0.50	1	1480.0	11.73
0.75	0	1947.0	13.47
1.00	0	2073.0	14.29
Random	15	569.5	11.64

Table 2. Performance of the different values of α for the proposed constructive method coupled with the local search after 10,000 iterations.

α	#B.	Cost	T(s)
0.00	29	470.9	11.13
0.25	22	474.0	18.44
0.50	17	479.6	24.41
0.75	16	480.2	26.61
1.00	17	484.3	27.00
Random	25	471.5	24.77

The comparison of Tables 1 and 2 evidences the contribution of the local search. In a pairwise comparison of rows from both tables, it can be seen that, for each value of α, the local search reaches better results not only in the number of times that the best value is obtained, but also in the quality of the solutions (see columns 2 and 3, respectively). Obviously, the CPU time is increased when the local search is run after the constructive process.

Given that the results of $\alpha = 0$ and α randomly chosen are very similar when the local search is run, both configurations will be selected for the next experiment.

In the following experiment we will assess the contribution of the proposed perturbation methods. To this aim, the perturbations have been run with different values for the perturbation size β, for both $\alpha = 0$ and α randomly chosen. Table 3 shows the results of this experiment, where the first column presents the two values of α considered in this experiment; the second column shows the three perturbation procedures, and the third column the values for the β parameter. The values for β have been determined experimentally taking into account a similar computational effort among the selected values. The remaining columns present the same results as in the previous experiment. To carry out this experimentation, we have run Algorithm 1 after one construction ($nc = 1$), which is the same for each instance in all the perturbation methods, hence performing a fair comparison among them. Besides, the value of np, which is the number of times the perturbation method is run, was set proportional to the number of occasional drivers. In particular, $np = 100 \cdot |K|$.

Table 3. Performance comparison among the proposed perturbation methods.

α	Perturbation	β	#B.	Cost	T(s)
0	*RandomMove*	10	23	443.4	4.43
		25	38	436.3	5.85
		50	**40**	**430.2**	6.19
		75	28	435.6	6.19
	RouteCost	1	1	513.5	2.78
		2	1	510.3	3.10
		3	1	514.8	3.42
	RandomDeassign	1	5	456.5	0.84
		2	10	452.6	2.04
		3	8	449.6	3.43
		5	10	448.4	5.71
Random	*RandomMove*	10	18	448.2	4.95
		25	29	436.5	7.50
		50	**35**	**433.3**	8.74
		75	28	435.6	8.47
	RouteCost	1	0	590.0	0.75
		2	0	590.2	0.82
		3	1	590.9	0.79
	RandomDeassign	1	17	451.8	3.67
		2	22	442.5	5.88
		3	19	441.4	7.37
		5	17	442.7	8.85

In view of the values shown in Table 3, it can be seen that the best results are obtained by the *RandomMove* method with $\beta = 50$, for both α values, and that the influence of the perturbation method over the final score is more important than the value of α.

Besides, the *RandomDeassign* method obtains competitive results, with a 2.5% difference with respect to the best perturbation configuration, while the *RouteCost* method obtains the worst results.

Finally, in order to take into account other possible algorithmic strategies, we designed a memetic approach [33] to tackle this problem. Here, the local search was combined with a genetic algorithm where the routes were encoded with a double chromosome for both the regular and the occasional drivers. The usual crossover and mutation operators were also implemented and several configurations were explored in relation with the execution of the local search step.

From those preliminary experiments, we show in Table 4 the results of the most relevant executions of the memetic approach, labeled as MA. In particular, the table shows the comparison of number of best results obtained by the ILS_M proposal, the exact approach from the state of the art (IP) and the memetic algorithm (MA) on a subset of small-sized instances.

Table 4. Comparison between ILS_M, the exact method (IP) and the memetic algorithm (MA).

|K|	ρ	ζ	ILS_M	**IP**	**MA**
13	0.2	1.1	6	6	1
		1.2	6	6	1
		1.3	6	6	2
		1.4	6	6	2
		1.5	6	6	1
25	0.2	1.1	6	6	1
		1.2	6	6	2
		1.3	6	6	2
		1.4	6	6	1
		1.5	5	6	2
			59	**60**	**15**

As it can be seen in the table, the memetic algorithm obtained poor results in relation with the two other proposals, while ILS_M was able to reach 59 out of the 60 optimal values. In addition, the execution time of MA was more than 50 times longer than the ILS_M approach. Therefore, we decided to omit the memetic approach from the final comparison.

4.2. Final Comparison

Once the ILS parameters have been studied in the previous section, we now proceed to compare our proposals with the state of the art, whose results will be labelled as $SOTA$. In brief, we propose a multi-start ILS algorithm, namely ILS; a straightforward parallel version of the multi-start ILS where the iterations of the algorithm are distributed among N threads, called ILS_P; and our proposed cooperative parallel ILS with migration of solutions, ILS_M. This experiment consists on running these algorithms on the whole set of 480 instances. Next, we describe the particular configuration selected for this proposal.

The number of iterations of ILS is 100, which corresponds to 100 constructions generated with $\alpha = 0$, and the perturbation method used was *RandomMove* with $\beta = 50$, since this configuration obtained the best results in the previous experimentation. The number of executions of the perturbation method was set to $np = 10 \cdot |K|$. This configuration is repeated for ILS_P given that this is a parallel implementation of ILS.

Regarding ILS_M, we take advantage of the cooperative policy by applying different configurations on each worker. In particular, we have considered $N = 4$ to be the number of cores used by the parallel versions of ILS, both ILS_P and ILS_M. Therefore, we consider 4 different configurations for ILS_M. In order to select these configurations we chose the four best configurations in terms of number of best solutions, as shown in Table 3: *RandomMove* with $\beta = 25$ ($\alpha = 0$ and $\alpha = Random$), and *RandomMove* with $\beta = 50$ ($\alpha = 0$ and $\alpha = Random$). The number of rounds was set to 2, making any solution go through every configuration twice, as explained in Section 3.4.

Table 5 shows the comparison of the proposed ILS algorithms with the state of the art for those instances with 13 and 25 occasional drivers. The results are summarized for each value of ζ, which represents the flexibility of the occasional drivers. For each algorithm (ILS, ILS_P and ILS_M) the averaged cost, the sum of best values (#B.), and the execution time in seconds (T(s)) are calculated. Since no information is given about execution time in [16], only the cost and the number of best results are reported for the SOTA. As it can be seen in the table, all our ILS proposals obtain the best result for all the instances, reaching the same average cost among them and improving the results of SOTA. Regarding the execution time, the fastest algorithm is ILS_P, which is a straightforward parallel implementation of ILS. ILS_M is slower than ILS_P because its execution time is determined by the slowest worker.

Table 5. Performance comparison for $|K| = 13$ and $|K| = 25$ occasional drivers.

$\|K\|$	ζ	ILS			ILS_P			ILS_M			SOTA	
		Cost	**#B.**	**T(s)**	**Cost**	**#B.**	**T(s)**	**Cost**	**#B.**	**T(s)**	**Cost**	**#B.**
13	1.1	242.38	6	8.67	242.38	6	1.84	242.38	6	3.13	242.53	5
	1.2	232.84	6	8.29	232.84	6	0.78	232.84	6	3.39	233.12	4
	1.3	230.28	6	8.08	230.28	6	7.00	230.28	6	3.30	230.58	4
	1.4	229.53	6	7.72	229.53	6	2.14	229.53	6	3.37	230.38	2
	1.5	222.27	6	7.40	222.27	6	1.72	222.27	6	3.13	223.43	1
		231.46	**30**	**8.03**	**231.46**	**30**	**2.69**	**231.46**	**30**	**21.26**	**232.01**	**16**
25	1.1	232.20	6	14.34	232.20	6	3.37	232.20	6	5.18	232.58	4
	1.2	227.96	6	14.32	227.96	6	3.38	227.96	6	5.75	228.23	5
	1.3	225.24	6	13.35	225.24	6	3.49	225.24	6	5.45	225.25	6
	1.4	216.07	6	11.84	216.07	6	4.07	216.07	6	5.18	216.08	6
	1.5	212.02	6	11.35	212.02	6	3.21	212.02	6	4.88	212.02	6
		222.70	**30**	**13.04**	**222.70**	**30**	**3.51**	**222.70**	**30**	**5.29**	**222.83**	**27**

For the medium size instances, where $|K| = 50$, the results are aggregated by the compensation rate (ρ) and the flexibility of the occasional drivers (ζ), as in [16]. Table 6 shows the results with the same indicators as in the small instances, but adding the relative percentage deviation from the best-known value (Gap). Looking at the results, we can point out that the average cost and the number of best results obtained by all the ILS proposals are better than the SOTA. If we focus on the number of best results (#B.), it can be seen that the basic ILS obtains practically the same results than the SOTA in this metric, however, the parallel collaborative scheme, ILS_M, obtains almost 50% more best results than the SOTA. Furthermore, regarding the gap, it can be seen that all the proposed algorithms obtain better relative deviations from the best-known values than the SOTA, specially the parallel collaborative scheme, ILS_M (0.21%). The execution time follows the same pattern as for the small instances, with ILS_P being the fastest algorithm and ILS_M the second one.

Finally, the results for the largest instances with 100 occasional drivers are shown in Table 7 in a similar fashion as the medium-size instances. As it can be seen, the performance gap between the ILS proposals and SOTA widens in terms of the number of best results. In particular, ILS_M reaches more than twice the number of best results than SOTA, while also improving the average cost and having less than half of the deviation. Note that the other two ILS proposals do not improve the average cost of SOTA by a small margin. However, both (ILS and ILS_P) improve the number of best results obtained. Regarding the average execution time, the results are similar than in the previous tables.

As a first conclusion from the results, we can affirm that the cooperative ILS_M proposal outperforms the basic ILS, the parallel ILS_P and the SOTA methods in terms of cost and, specially, in terms of number of best results found. As previously mentioned, the ILS_M method is slower than ILS_P as its computation time is limited by the slowest worker. For the sake of space we have omitted the detailed results of all instances. However, we will make them publicly available at http://grafo.etsii.urjc.es/ (accessed on 16 January 2021).

In order to statistically assess the behavior of the algorithms considered in this work, we carried out the Bayesian performance analysis for comparing multiple algorithms over multiple instances simultaneously described in [34,35]. This analysis considers the experimental results as rankings of algorithms, and on the basis of a probability distribution defined on the space of rankings, it computes the expected probability of each algorithm being the best among the compared ones. Not limited to that, it also assesses the uncertainty related to the estimation in the form of credible intervals. These intervals are computed using Bayesian statistics and they estimate the most likely values of the unknown parameter to lie within the interval given a certain probability.

Table 6. Performance comparison for $|K| = 50$ occasional drivers.

ρ	ζ	ILS				ILS_P				ILS_M				SOTA		
		Cost	**Gap**	**#B.**	**T(s)**	**Cost**	**Gap**	**#B.**	**T(s)**	**Cost**	**Gap**	**#B.**	**T(s)**	**Cost**	**Gap**	**#B.**
0.05	1.1	539.7853	1.40%	2	419.5918	540.5165	1.53%	0	101.7118	532.3479	0.00%	4	170.3358	537.7986	1.02%	1
	1.2	503.2001	0.80%	3	389.4178	500.6505	0.29%	3	64.05583	503.625	0.88%	2	203.561	499.225	0.00%	2
	1.3	486.9723	0.26%	1	386.9173	488.1669	0.50%	4	95.45933	485.7328	0.00%	3	195.2685	491.2126	1.13%	0
	1.4	471.8444	1.01%	3	382.2975	467.1162	0.00%	4	103.3242	470.8017	0.79%	1	193.2252	473.574	1.38%	1
	1.5	453.8147	0.00%	3	383.0937	455.7323	0.42%	0	55.57933	454.9244	0.24%	2	191.0568	456.8082	0.66%	2
0.1	1.1	584.9547	0.00%	1	466.8708	585.692	0.13%	1	96.67783	585.4194	0.08%	2	233.156	587.1946	0.38%	2
	1.2	551.9803	0.47%	2	430.2182	553.8638	0.81%	3	79.56183	553.4574	0.74%	3	231.1562	549.3987	0.00%	2
	1.3	541.1348	0.00%	3	420.7238	547.067	1.10%	4	85.77933	546.0515	0.91%	3	222.7265	544.3012	0.59%	1
	1.4	533.3659	0.59%	1	414.1407	535.6826	1.03%	3	100.1178	537.2698	1.33%	2	221.526	530.2156	0.00%	3
	1.5	524.9017	1.55%	3	415.2357	517.1454	0.05%	3	117.9817	519.1073	0.43%	1	217.0753	516.8894	0.00%	3
0.2	1.1	663.5926	0.16%	3	595.0698	663.7559	0.18%	2	141.5705	662.5436	0.00%	3	315.4683	666.3268	0.57%	3
	1.2	643.8584	0.38%	2	568.4518	641.8753	0.07%	3	87.92133	641.4362	0.00%	5	326.2498	642.3602	0.14%	1
	1.3	641.0553	0.79%	2	562.1503	639.2008	0.50%	3	122.836	639.2933	0.51%	2	329.638	636.0185	0.00%	3
	1.4	634.2417	0.24%	3	544.6653	633.5266	0.13%	3	69.31633	632.6958	0.00%	3	324.1843	634.2424	0.24%	3
	1.5	626.0921	0.00%	3	535.832	626.2378	0.02%	3	81.129	627.0634	0.16%	2	320.1565	628.7249	0.42%	2
1.2	1.1	548.2522	0.05%	1	419.0178	548.4516	0.08%	2	141.4412	547.9879	0.00%	2	253.8163	556.4641	1.55%	1
	1.2	532.7871	0.00%	2	427.1283	533.5399	0.14%	0	51.02833	533.2159	0.08%	2	237.4145	541.4967	1.63%	2
	1.3	525.7513	0.16%	2	435.0688	532.1804	1.38%	1	109.7833	524.9333	0.00%	2	243.4765	534.9408	1.91%	1
	1.4	526.8243	0.25%	0	443.8138	527.0245	0.29%	3	88.965	525.5251	0.00%	2	248.429	534.9803	1.80%	1
	1.5	530.9357	0.91%	0	453.4277	526.1489	0.00%	4	66.63233	526.6752	0.10%	3	258.1862	531.9392	1.10%	1
1.4	1.1	552.3821	0.21%	1	430.4313	551.2252	0.00%	3	91.93033	551.5696	0.06%	2	234.8208	559.322	1.47%	2
	1.2	538.8656	0.17%	1	432.7217	538.7884	0.15%	1	57.576	537.9606	0.00%	2	240.0383	546.494	1.59%	2
	1.3	536.8756	0.47%	1	441.8725	534.9065	0.10%	3	110.9935	534.3706	0.00%	2	249.0508	543.1721	1.65%	1
	1.4	536.7697	0.86%	1	450.4062	536.4604	0.80%	1	108.433	532.1984	0.00%	3	255.0165	540.0234	1.47%	2
	1.5	536.2398	0.51%	1	459.4875	535.8174	0.43%	1	132.6498	533.5195	0.00%	3	265.1078	539.6146	1.14%	2
1.6	1.1	554.5638	0.98%	1	427.4273	554.5725	0.98%	2	73.339	549.1668	0.00%	4	245.174	563.0623	2.53%	0
	1.2	543.4267	0.16%	3	442.693	544.2657	0.32%	0	120.5182	542.5449	0.00%	3	245.2358	551.6283	1.67%	1
	1.3	543.3848	0.17%	1	451.8478	544.4085	0.36%	0	297.7438	542.4653	0.00%	3	254.8628	547.8221	0.99%	2
	1.4	541.4304	0.13%	0	460.8212	540.9699	0.04%	2	517.5568	540.7474	0.00%	2	266.4755	546.7086	1.10%	2
	1.5	543.274	0.36%	2	468.7355	541.3017	0.00%	2	792.5418	541.4565	0.03%	1	276.5442	547.0722	1.07%	1
		549.75	**0.43%**	**52**	**451.99**	**549.54**	**0.39%**	**64**	**138.81**	**548.54**	**0.21%**	**74**	**248.95**	**552.63**	**0.97%**	**50**

Table 7. Performance comparison for $|K| = 100$ occasional drivers.

ρ	ζ	ILS				ILS_P				ILS_M				SOTA		
		Cost	**Gap**	**#B.**	**T(s)**	**Cost**	**Gap**	**#B.**	**T(s)**	**Cost**	**Gap**	**#B.**	**T(s)**	**Cost**	**Gap**	**#B.**
0.05	1.1	446.8542	0.50%	3	461.8845	446.7632	0.48%	5	50.84183	444.628	0.00%	5	176.1205	445.5205	0.20%	4
	1.2	393.4636	0.01%	5	347.7535	394.1906	0.20%	4	124.0983	393.4137	0.00%	6	170.5258	397.6275	1.07%	2
	1.3	309.0867	0.00%	6	220.556	309.0867	0.00%	6	76.62167	309.2135	0.04%	5	127.8515	309.1152	0.01%	5
	1.4	291.847	0.09%	5	160.219	291.8418	0.09%	5	99.157	291.8761	0.10%	5	95.24367	291.5823	0.00%	4
	1.5	256.6883	1.79%	5	128.2215	256.6883	1.79%	5	48.07133	252.8473	0.27%	5	79.11217	252.168	0.00%	6
0.1	1.1	523.0534	1.22%	3	654.3707	519.6255	0.56%	3	120.4445	516.7501	0.00%	6	277.8033	518.4761	0.33%	2
	1.2	479.7269	0.31%	4	539.1515	481.4061	0.66%	4	157.8802	478.2522	0.00%	5	274.8117	482.0614	0.80%	4
	1.3	427.8776	2.86%	3	462.71	426.3933	2.50%	3	129.64	424.1554	1.96%	4	232.155	415.9928	0.00%	3
	1.4	402.368	0.16%	4	415.5207	402.368	0.16%	4	95.69033	401.7055	0.00%	6	211.4222	402.6224	0.23%	2
	1.5	384.158	1.88%	4	388.4473	397.081	5.31%	3	89.23267	388.1649	2.94%	4	191.0957	377.0738	0.00%	4
0.2	1.1	639.8737	0.00%	3	1006.771	642.5748	0.42%	4	161.6953	642.3011	0.38%	3	565.708	641.3445	0.23%	3
	1.2	623.6678	0.92%	4	964.2745	622.5619	0.74%	5	166.8772	623.7514	0.93%	4	591.0685	618.008	0.00%	2
	1.3	605.705	0.91%	4	873.0052	605.7514	0.92%	4	158.8033	602.0615	0.30%	4	546.6782	600.2485	0.00%	3
	1.4	596.3077	0.73%	4	845.7318	594.0314	0.35%	4	121.6118	594.0179	0.35%	4	540.0835	591.9684	0.00%	4
	1.5	591.5529	1.30%	3	830.2405	590.8163	1.17%	3	57.886	589.4598	0.94%	3	529.7783	583.9739	0.00%	3
1.2	1.1	453.5543	0.56%	3	532.8617	451.9424	0.21%	4	124.8337	451.0063	0.00%	5	329.6615	453.9328	0.65%	1
	1.2	437.11	0.56%	4	516.6463	437.2956	0.61%	2	150.8505	434.6544	0.00%	5	248.1792	439.1365	1.03%	0
	1.3	413.5353	0.61%	2	470.4215	414.8368	0.93%	2	44.55867	411.0114	0.00%	6	235.5038	418.7916	1.89%	0
	1.4	416.3125	1.60%	2	475.7985	413.6793	0.96%	1	127.474	409.7569	0.00%	6	255.2198	417.0191	1.77%	0
	1.5	417.7203	1.80%	1	489.0567	413.7277	0.83%	3	92.63217	410.3363	0.00%	4	263.5218	414.6062	1.04%	1
1.4	1.1	457.0456	0.05%	3	537.4583	456.877	0.02%	3	128.7593	456.8032	0.00%	5	283.0257	459.9442	0.69%	1
	1.2	445.1086	0.19%	2	531.8792	445.9219	0.38%	1	128.3608	444.2435	0.00%	5	265.7002	448.9299	1.05%	2
	1.3	428.7152	0.66%	4	509.4325	430.9109	1.17%	1	109.2697	425.9209	0.00%	2	267.0003	435.4451	2.24%	1
	1.4	431.734	0.91%	2	510.7673	428.8154	0.23%	2	110.4097	427.9069	0.01%	4	276.2122	427.8523	0.00%	2
	1.5	430.9797	0.62%	2	528.5645	430.8396	0.58%	1	64.2445	428.3419	0.00%	4	292.21	432.2809	0.92%	2
1.6	1.1	461.8792	0.01%	3	558.2377	463.1208	0.28%	2	115.4152	461.8351	0.00%	4	294.3458	465.8242	0.86%	0
	1.2	453.4498	0.08%	2	572.0842	453.1118	0.01%	3	78.56083	453.0686	0.00%	4	283.187	457.9317	1.07%	1
	1.3	442.0516	0.25%	2	555.9252	440.9349	0.00%	3	66.0465	441.321	0.09%	3	295.9443	449.6249	1.97%	1
	1.4	441.1734	0.18%	1	555.8278	440.3888	0.00%	4	109.6753	440.6528	0.06%	3	307.37	443.2396	0.65%	1
	1.5	441.7029	0.27%	2	574.9573	441.3502	0.19%	3	88.07833	440.5029	0.00%	4	322.1748	443.3912	0.66%	1
		451.48	**0.70%**	**95**	**540.63**	**451.50**	**0.72%**	**97**	**106.59**	**449.67**	**0.28%**	**133**	**294.29**	**451.19**	**0.65%**	**65**

Figure 7 shows the credible intervals (5% and 95% quantiles) and the expected probability of winning for each different implementation of the proposed algorithms and the state-of-the-art method (SOTA) after the joint analysis of the 480 instances. We will refer to the term *winning* when the algorithm is able to find the best solution in relation to the other methods in the comparison. As seen in the figure, SOTA is the algorithm with least chances for being the winner, with an expected probability of 0.127 of obtaining the best solution. Besides, the probability of ILS and ILS_P is quite similar (0.243 and 0.276 respectively), with overlapped credible intervals. This result proves that a straightforward parallelization has a small contribution to the quality reached by the algorithm, and the main advantage is the savings in computation time. However, both proposals are better than SOTA since their probabilities of obtaining better solutions than SOTA are higher. Moreover, the expected probability of ILS_M is the highest, reaching a value of 0.354, showing a credible interval that is not overlapped with any other. In summary, ILS_M is statistically different from all the other algorithms, and it will reach the best solutions in almost 36% of the instances. The observed length for the intervals in Figure 7 points out that the estimations for SOTA and ILS_M permit to draw solid conclusions, while for the case of ILS_P and ILS, due to the overlapping of the intervals, both algorithms have similar probability for being the winners.

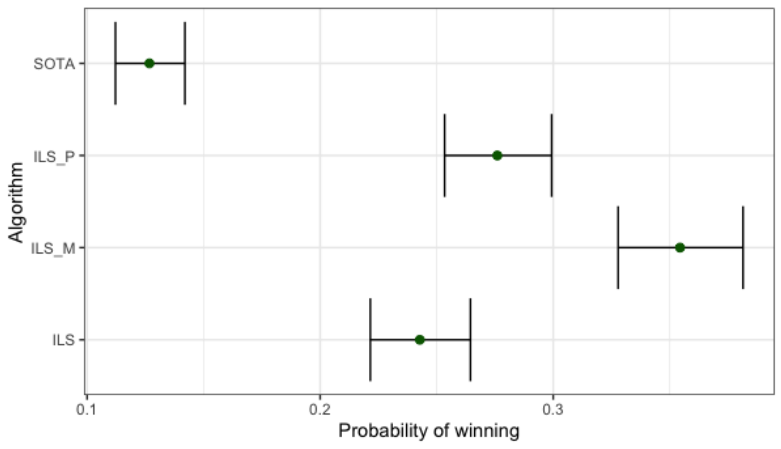


Figure 7. Credible intervals (5% and 95% quantiles) and expected probability of winning for the proposed Iterated Local Search (ILS) algorithms and the state of the art (SOTA).

Therefore, this statistical analysis proves that, on the one hand, all our ILS proposals obtain better results than the state-of-the-art method and, on the other hand, the proposed cooperative scheme makes a significant difference in relation to the other ILS proposals.

5. Conclusions

Sustainable logistics require the combination of the traditional business logistics and crowd logistics. However, efficient optimization algorithms are required in order to merge these approaches into corporate information systems.

In this work we propose an efficient optimization algorithm based on the Iterated Local Search method which we have assessed on the Vehicle Routing Problem with Occasional Drivers (VRPOD). This problem models realistic situations appearing in the transportation of goods for delivery companies in which crowdshipping, a sustainable means of transport, is feasible, taking into account two different compensation schemes.

In particular, the three proposed implementations of the ILS are able to overcome the results obtained by the state of the art. The ILS design proposed in this paper includes a greedy randomized constructive method to build initial solutions and a local search which explores an extended neighborhood formed by the combination of neighborhoods generated by five different moves. In addition, three perturbation strategies have been proposed for this problem. Moreover, a parallel cooperation scheme has been designed for the ILS proposal. The computational experiments evidence the effectiveness of our algorithm given that it is able to attain all the optimal values when they are known. Besides, it obtains better results than the state-of-the-art method spending a competitive execution time. A statistical assessment of the proposed algorithms performance has been also included, measuring the differences between the ILS methods and the state of the art. In the light of the computational results, we can state that the three different implementations based on the ILS methodology are able to improve the state of the art for the small instances (instances with 13 and 25 occasional drivers) in a few seconds, finding 17 new best-known solutions. A similar behaviour can be observed for the medium-size instances (50 occasional drivers), finding 130 out of 180 new best-known solutions. Finally, in the large instances (those with 100 occasional drivers), the cooperative parallel ILS with solution migrations, ILS_M, not only reduces the average cost value but also reaches 68 out of 180 new best-known solutions. Furthermore, considering all the three different implementations based on the ILS methodology 115 in total new best-known solutions. To prove the hypothesis that the ILS_M is the best proposal, a statistical analysis has been included where all the algorithms have been compared. As a conclusion of this analysis, the three different implementations based on the ILS algorithm outperform the state of the art, being ILS_M the best one among all the studied alternatives.

Therefore, since the previous results are improved by including occasional drivers with our proposal, we can state that our method is able to optimize the last-mile logistics, as recommended in [14]. Hence, since the new routes have smaller costs, the sustainability of the companies is favored. Besides, the reduced computation times of ILS_M allows the inclusion of our method into corporate information systems.

Future research directions could include different compensation schemes profitable not only for the company but also for the occasional driver, which lead us to a multi-objective optimization problem since both objectives are clearly in conflict. Furthermore, another interesting future line would be to include the possibility of allowing more than one delivery to every occasional driver or even sustainability features included also under a multi-objective approach to show the trade-off among the different objective functions. In addition, the use of more detailed instances with information about the type of vehicles used by the occasional drives will allow to obtain sustainability measures such as the carbon footprint of a route. Finally, in order to study more realistic scenarios, new instances with stochastic modeling of the demand or the travelling times could be defined, as suggested in [36]. Of course, adding new features to the considered problem would lead us to adapt the ILS methodology and check the robustness of our algorithm since new constraints are incorporated.

Author Contributions: Conceptualization, R.M.-S., A.D.L.-S. and J.M.C.; Data curation, R.M.-S., A.D.L.-S. and J.M.C.; Formal analysis, A.D.L.-S.; Funding acquisition, M.L.D.-J.; Methodology, R.M.-S., A.D.L.-S. and J.M.C.; Project administration, M.L.D.-J.; Software, R.M.-S. and J.M.C.; Supervision, J.M.C.; Writing—original draft, R.M.-S., A.D.L.-S. and J.M.C.; Writing—review & editing, R.M.-S., A.D.L.-S., M.L.D.-J. and J.M.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been partially supported by the Spanish Ministerio de Ciencia, Innovación y Universidades (MCIU/AEI/FEDER, UE) under grant ref. PGC2018-095322-B-C2; Comunidad de Madrid y Fondos Estructurales de la Unión Europea under grant ref. S2018/TCS-4566 and the Junta de Andalucía, FEDER-UPO Research & Development Call, under grant ref. UPO-1263769.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Detailed results and instances will be publicly available at http://grafo.etsii.urjc.es/ (accessed on 16 January 2021).

Acknowledgments: The authors wish to express their gratitude to C. Archetti, M. Savelsbergh and G. Speranza for their collaboration in the provision of all their previous results.

Conflicts of Interest: The authors declare no conflict of interest.

References

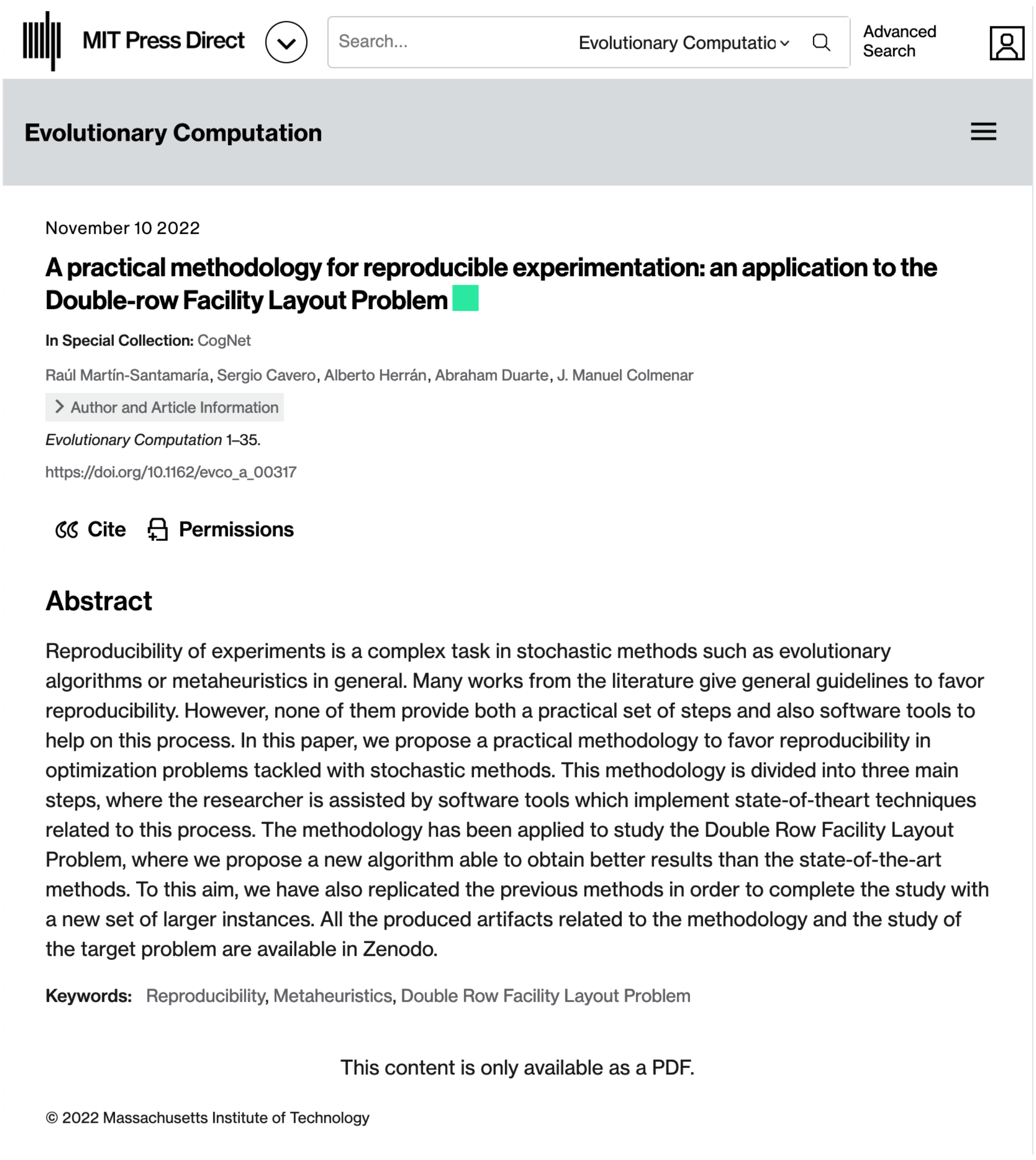
1. Allen, J.; Piecyk, M.; Piotrowska, M.; McLeod, F.; Cherrett, T.; Ghali, K.; Nguyen, T.; Bektas, T.; Bates, O.; Friday, A.; et al. Understanding the impact of e-commerce on last-mile light goods vehicle activity in urban areas: The case of London. *Transp. Res. Part Transp. Environ.* **2018**, *61*, 325–338. [CrossRef]
2. Jia, Y.H.; Chen, W.N.; Gu, T.; Zhang, H.; Yuan, H.; Lin, Y.; Yu, W.J.; Zhang, J. A dynamic logistic dispatching system with set-based particle swarm optimization. *IEEE Trans. Syst. Man Cybern. Syst.* **2017**, *48*, 1607–1621. [CrossRef]
3. Qin, G.; Tao, F.; Li, L. A vehicle routing optimization problem for cold chain logistics considering customer satisfaction and carbon emissions. *Int. J. Environ. Res. Public Health* **2019**, *16*, 576. [CrossRef]
4. Gevaers, R.; Van de Voorde, E.; Vanelslander, T. Characteristics of innovations in last-mile logistics-using best practices, case studies and making the link with green and sustainable logistics. *Assoc. Eur. Transp. Contrib.* **2009** 1–21.
5. Sampaio, A.; Savelsbergh, M.; Veelenturf, L.; Van Woensel, T. Chapter 15: Crowd-Based City Logistics. In *Sustainable Transportation and Smart Logistics*; Faulin, J., Grasman, S.E., Juan, A.A., Hirsch, P., Eds.; Elsevier: Amsterdam, The Netherlands, 2019; pp. 381–400.
6. Carbone, V.; Rouquet, A.; Roussat, C. The Rise of Crowd Logistics: A New Way to Co-Create Logistics Value. *J. Bus. Logist.* **2017**, *38*, 238–252. [CrossRef]
7. Devari, A.; Nikolaev, A.G.; He, Q. Crowdsourcing the last mile delivery of online orders by exploiting the social networks of retail store customers. *Transp. Res. Part Logist. Transp. Rev.* **2017**, *105*, 105–122. [CrossRef]
8. Botsman, R. Crowdshipping: Using the crowd to transform delivery. *AFR Boss Mag.* **2014**.
9. Mckinnon, A. Crowdshipping: A Communal Approach to Reducing Urban Traffic Levels? 2016. Available online: https://www.alanmckinnon.co.uk/story_layout.html?IDX=714&b=56 (accessed on 6 January 2021)
10. Guo, X.; Jaramillo, Y.J.L.; Bloemhof-Ruwaard, J.; Claassen, G. On integrating crowdsourced delivery in last-mile logistics: A simulation study to quantify its feasibility. *J. Clean. Prod.* **2019**, *241*, 118365. [CrossRef]
11. Simoni, M.D.; Marcucci, E.; Gatta, V.; Claudel, C.G. Potential last-mile impacts of crowdshipping services: A simulation-based evaluation. *Transportation* **2020**, *47*, 1933–1954. [CrossRef]
12. Toth, P.; Vigo, D. *Vehicle Routing: Problems, Methods, and Applications*; SIAM: Philadelphia, PA, USA, 2014.
13. Braekers, K.; Ramaekers, K.; Van Nieuwenhuyse, I. The vehicle routing problem: State of the art classification and review. *Comput. Ind. Eng.* **2016**, *99*, 300–313. [CrossRef]
14. Ranieri, L.; Digiesi, S.; Silvestri, B.; Roccotelli, M. A review of last mile logistics innovations in an externalities cost reduction vision. *Sustainability* **2018**, *10*, 782. [CrossRef]
15. Wang, Y.; Zhang, D.; Liu, Q.; Shen, F.; Lee, L.H. Towards enhancing the last-mile delivery: An effective crowd-tasking model with scalable solutions. *Transp. Res. Part Logist. Transp. Rev.* **2016**, *93*, 279–293. [CrossRef]
16. Archetti, C.; Savelsbergh, M.; Speranza, M.G. The Vehicle Routing Problem with Occasional Drivers. *Eur. J. Oper. Res.* **2016**, *254*, 472–480. [CrossRef]
17. Dahle, L.; Andersson, H.; Christiansen, M. *The Vehicle Routing Problem with Dynamic Occasional Drivers*; Computational Logistics; Bektaş, T., Coniglio, S., Martinez-Sykora, A., Voß, S., Eds.; Springer International Publishing: Cham, Switzerland, 2017; pp. 49–63.
18. Macrina, G.; Di Puglia Pugliese, L.; Guerriero, F.; Laganà, D. The Vehicle Routing Problem with Occasional Drivers and Time Windows. In *Optimization and Decision Science: Methodologies and Applications*; Sforza, A., Sterle, C., Eds.; Springer International Publishing: Cham, Switzerland, 2017; pp. 577–587.
19. Gdowska, K.; Viana, A.; Pedroso, J.P. Stochastic last-mile delivery with crowdshipping. *Transp. Res. Procedia* **2018**, *30*, 90–100. [CrossRef]
20. Arslan, A.M.; Agatz, N.; Kroon, L.; Zuidwijk, R. Crowdsourced Delivery: A Dynamic Pickup and Delivery Problem with Ad-Hoc Drivers. *Transp. Sci.* **2019**, *53*, 222–235. [CrossRef]
21. Behrend, M.; Meisel, F.; Fagerholt, K.; Andersson, H. An exact solution method for the capacitated item-sharing and crowdshipping problem. *Eur. J. Oper. Res.* **2019**, *279*, 589–604. [CrossRef]
22. Brandão, J. A deterministic iterated local search algorithm for the vehicle routing problem with backhauls. *TOP* **2016**, *24*, 445–465. [CrossRef]
23. Cattaruzza, D.; Absi, N.; Feillet, D.; Vigo, D. An iterated local search for the multi-commodity multi-trip vehicle routing problem with time windows. *Comput. Oper. Res.* **2014**, *51*, 257–267. [CrossRef]
24. Silva, M.M.; Subramanian, A.; Ochi, L.S. An iterated local search heuristic for the split delivery vehicle routing problem. *Comput. Oper. Res.* **2015**, *53*, 234–249. [CrossRef]
25. Lourenço, H.R.; Martin, O.C.; Stützle, T. Iterated Local Search. In *Handbook of Metaheuristics*; Glover, F., Kochenberger, G.A., Eds; Springer: Boston, MA, USA, 2003; pp. 320–353.
26. Feo, T.A.; Resende, M.G.C. A probabilistic heuristic for a computationally difficult set covering problem. *Oper. Res. Lett.* **1989**, *8*, 67–71. [CrossRef]
27. Feo, T.A.; Resende, M.G.C.; Smith, S.H. A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set. *Oper. Res.* **1994**, *42*, 860–878. [CrossRef]
28. Croes, G.A. A Method for Solving Traveling-Salesman Problems. *Oper. Res.* **1958**, *6*, 791–812. [CrossRef]

29. Faulin, J.; Juan, A.A. The ALGACEA-1 method for the capacitated vehicle routing problem. *Int. Trans. Oper. Res.* **2008**, *15*, 599–621. [CrossRef]
30. Buxey, G.M. The Vehicle Scheduling Problem and Monte Carlo Simulation. *J. Oper. Res. Soc.* **1979**, *30*, 563–573. [CrossRef]
31. Crainic, T.; Gendreau, M.; Hansen, P.; Mladenovic, N. Cooperative Parallel Variable Neighborhood Search for the p-Median. *J. Heuristics* **2004**, *10*, 293–314. [CrossRef]
32. Herrán, A.; Colmenar, J.M.; Martí, R.; Duarte, A. A parallel variable neighborhood search approach for the obnoxious p-median problem. *Int. Trans. Oper. Res.* **2020**, *27*, 336–360. [CrossRef]
33. Neri, F.; Cotta, C.; Moscato, P. *Handbook of Memetic Algorithms*; Springer: Cham, Switzerland, 2011.
34. Calvo, B.; Ceberio, J.; Lozano, J.A. Bayesian Inference for Algorithm Ranking Analysis. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*; ACM: New York, NY, USA, 2018; pp. 324–325.
35. Calvo, B.; Shir, O.M.; Ceberio, J.; Doerr, C.; Wang, H.; Bäck, T.; Lozano, J.A. Bayesian Performance Analysis for Black-box Optimization Benchmarking. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*; ACM: New York, NY, USA, 2019; pp. 1789–1797.
36. Juan, A.A.; David Kelton, W.; Currie, C.S.M.; Faulin, J. Simheuristics Applications: Dealing With Uncertainty In Logistics, Transportation, and Other Supply Chain Areas. In Proceedings of the 2018 Winter Simulation Conference (WSC), Gothenburg, Sweden, 9–12 December 2018; pp. 3048–3059.

Chapter 8

A practical methodology for reproducible experimentation: an application to the Double-row Facility Layout Problem

Title	A practical methodology for reproducible experimentation: an application to the Double-row Facility Layout Problem
Authors	Raúl Martín-Santamaría, Sergio Cavero, Alberto Herrán, Abraham Duarte and José Manuel Colmenar
Publication date	2022 (Early Access)
Journal	Evolutionary Computation
Publisher	MIT Press
ISBN/ISSN	1063-6560
Impact Factor	4.766 (2021)
Rank by Impact Factor	16/110 (Q1, Computer Science, Theory & Methods)
DOI	`https://doi.org/10.1162/evco_a_00317`



Evolutionary Computation

November 10 2022

A practical methodology for reproducible experimentation: an application to the Double-row Facility Layout Problem 🟩

In Special Collection: CogNet

Raúl Martín-Santamaría, Sergio Cavero, Alberto Herrán, Abraham Duarte, J. Manuel Colmenar

> Author and Article Information

Evolutionary Computation 1–35.

https://doi.org/10.1162/evco_a_00317

Cite Permissions

Abstract

Reproducibility of experiments is a complex task in stochastic methods such as evolutionary algorithms or metaheuristics in general. Many works from the literature give general guidelines to favor reproducibility. However, none of them provide both a practical set of steps and also software tools to help on this process. In this paper, we propose a practical methodology to favor reproducibility in optimization problems tackled with stochastic methods. This methodology is divided into three main steps, where the researcher is assisted by software tools which implement state-of-theart techniques related to this process. The methodology has been applied to study the Double Row Facility Layout Problem, where we propose a new algorithm able to obtain better results than the state-of-the-art methods. To this aim, we have also replicated the previous methods in order to complete the study with a new set of larger instances. All the produced artifacts related to the methodology and the study of the target problem are available in Zenodo.

Keywords: Reproducibility, Metaheuristics, Double Row Facility Layout Problem

This content is only available as a PDF.

A practical methodology for reproducible experimentation: an application to the Double-row Facility Layout Problem

Raúl Martín-Santamaría raul.martin@urjc.es
Department of Computer Science and Statistics, Universidad Rey Juan Carlos, Móstoles, 28933, Spain

Sergio Cavero sergio.cavero@urjc.es
Department of Computer Science and Statistics, Universidad Rey Juan Carlos, Móstoles, 28933, Spain

Alberto Herrán alberto.herran@urjc.es
Department of Computer Science and Statistics, Universidad Rey Juan Carlos, Móstoles, 28933, Spain

Abraham Duarte abraham.duarte@urjc.es
Department of Computer Science and Statistics, Universidad Rey Juan Carlos, Móstoles, 28933, Spain

J. Manuel Colmenar josemanuel.colmenar@urjc.es
Department of Computer Science and Statistics, Universidad Rey Juan Carlos, Móstoles, 28933, Spain

Abstract

Reproducibility of experiments is a complex task in stochastic methods such as evolutionary algorithms or metaheuristics in general. Many works from the literature give general guidelines to favor reproducibility. However, none of them provide both a practical set of steps and also software tools to help on this process. In this paper, we propose a practical methodology to favor reproducibility in optimization problems tackled with stochastic methods. This methodology is divided into three main steps, where the researcher is assisted by software tools which implement state-of-the-art techniques related to this process. The methodology has been applied to study the Double Row Facility Layout Problem, where we propose a new algorithm able to obtain better results than the state-of-the-art methods. To this aim, we have also replicated the previous methods in order to complete the study with a new set of larger instances. All the produced artifacts related to the methodology and the study of the target problem are available in Zenodo.

Keywords

Reproducibility, Metaheuristics, Double Row Facility Layout Problem.

1 Introduction

There exists a growing concern in the research community about the lack of methodological frameworks to compare stochastic algorithms. The list of problems ranges from the election of instances (Bartz-Beielstein et al., 2020) to ensuring the experimental reproducibility (Kendall et al., 2016). Evolutionary computation (Bäck et al., 1997) in particular, and metaheuristics (Glover and Kochenberger, 2006) in general, are among the most renowned stochastic algorithms. In this kind of procedures, results are supported by empirical studies. Therefore, the ability to reach similar results (both, in quality and computing time) by repeating an experiment performed by other researchers is the only way in which the research community can reach a consensus about an empirical finding. Despite a set of guidelines for Optimization Studies in Software Engineering exists as part of the ACM SGSOFT Empirical Standards effort (Ralph et al., 2020), many scientific works are not reproducible at all (Baker, 2016). Moreover, reproducibility issues do not only affect researchers when reproducing experiments conducted by other researchers, but also when reproducing their own previous experiments.

In this paper, we mainly focus on reproducibility in the context of stochastic procedures. It is worth mentioning that there does not exist a well established terminology in this matter. Indeed, terms such as reproducibility and replicability are used almost as synonymous (López-Ibáñez et al., 2021). We refer the reader to Plesser (2018) for further details about the evolution of the terminology in this area.

The first studies of reproducibility can be traced back to the nineties (Moscato and Norman, 1992). In the same line, Gent et al. (1997) claimed that the empirical study of algorithms is a relatively "immature" field, proposing (colloquially) some "thing to do" and "thing to not do" in order to ensure the experimental reproducibility. Later, Johnson (2002) provided a useful guide to computer scientists about how experimental analysis of algorithms can be performed and, then, described. A similar analysis was conducted in Eiben and Jelasity (2002) but focused on evolutionary algorithms. The Association of Computing Machinery (ACM) has recently specified different degrees of reproducibility (ACM, 2021) by considering the concepts of *artifact* and *measurement*. Specifically, an artifact is defined as a digital object that was either created by researchers to be used as part of the study or generated by the experiment itself. Examples of artifacts in the context of stochastic procedures would be source/executable code of an algorithm or data/code required to fully specify benchmark instances. A measurement is introduced as an analogy to physical experiments. In the context of stochastic procedures, it represents the raw data (objective function values, runtimes, etc.) that results from an experiment. Notice that depending on the level of abstraction being studied, sometimes reporting aggregated results is more adequate. For example, summary statistics such as means and standard errors, computational effort evaluated as cycles, function evaluations, or iteration counters.

Based on these definitions, ACM introduced three new concepts: repeatability, reproducibility, and replicability (ACM, 2021). However, more recently, a redefinition of this classification was proposed in López-Ibáñez et al. (2021), also adding the concept of generalizability. This new classification is based on the definition of fixed and random factors, which are determined by the researcher and given by elements like random seeds, respectively. The Turing Way project (Arnold et al., 2019) introduced a complementary approach. Specifically, it considers reproducibility (same analysis performed on the same dataset consistently produces the same answer), replicability (same analysis performed on different datasets produces qualitatively similar answers), robustness (different analysis applied to the same dataset produces a qualitatively similar answer to the same question), and generalizability (the combination of replicability and robustness). This classification has been mainly adopted in the Machine Learning community (Pineau et al., 2021). An alternative approach is introduced in Stodden et al. (2014), which relates to the availability of the code, data and all the details of the implementation and experimental setup that allow obtaining the published result. It also considers statistical reproducibility, which is concerned with validating the results of repeated experiments by means of statistical assessments.

There exists a vast literature (see López-Ibáñez et al. (2021) for a recent survey) asking for general guidelines that aim to ensure, assess, and encourage the reproducibility of experiments. For instance, by publishing permanently accessible, complete, and useful artifacts. Unfortunately, this is particularly difficult in stochastic procedures, since one must deal not only with differences on hardware and/or software for reproducing experiments, but also with algorithm randomness. Replicating the exact conditions of previously reported experiments might be impossible, even for those researchers that originally proposed the corresponding experiment. This situation arises when, for example, the original hardware is not available anymore, the version of some software libraries is not specified, or some sources of randomness are not repeatable, among others. In that case, repeatability and reproducibility are unreachable. Replicability may still be achievable, but, unfortunately, it is not an easy task since replicability requires high-level descriptions of artifacts with enough detail to enable their independent implementation and a careful selection of measurements with their stated precision and confidence levels. Only if those prerequisites are fulfilled, other researchers can unequivocally conclude if the replication of an experiment is confirmed or not.

In this paper, we propose a methodological approach to increase the reproducibility of empirical results. It is based on an automated characterization and selection of test instances and an automated configuration of the proposed algorithm. In addition, source code, instances, and executable artifacts are publicly available in Zenodo. Being this the main contribution, we validate the proposed methodology with a well-known combinatorial optimization problem. Specifically, we propose a new metaheuristic method for the Double Row Facility Layout Problem which is able to obtain better results than the state-of-the-art algorithms using shorter computation times. In particular, the proposal obtains all best-known values in a fraction of the time required by previous methods. The new algorithm is not an outcome of the methodology, but the reproducibility of the conducted experimentation is.

The rest of the paper is organized as follows. The proposed methodology for reproducibility is described in Section 2. The definition of the target problem and the algorithmic proposals are detailed in Sections 3 and 4, respectively. The experimental results and the application of the proposed methodology to the target problem are explained in 5, and a thorough description of the provided artifacts is given in Section 6. Finally, conclusions and future work are drawn in Section 7.

2 Generic methodology for reproducibility

The main goal of this paper is to encourage reproducibility efforts for optimization problems being solved with metaheuristics. In order to do so, we propose an empirical methodology based on minimizing the number of decisions taken by researchers (i.e., automatizing the majority of them). It is worth mentioning that this methodology is suitable not only for any evolutionary algorithm, but also for metaheuristic procedures in general.

The diagram in Figure 1 schematically illustrates the proposed methodology, where three main steps have been highlighted: benchmark instance selection, parameter tuning, and artifact generation. This approach starts by receiving the full set of instances that will be tackled in the considered optimization problem, returning the set of artifacts ready to be used by the research community. As shown in Figure 1, this methodology is composed of three main steps. In the first one, instances are classified

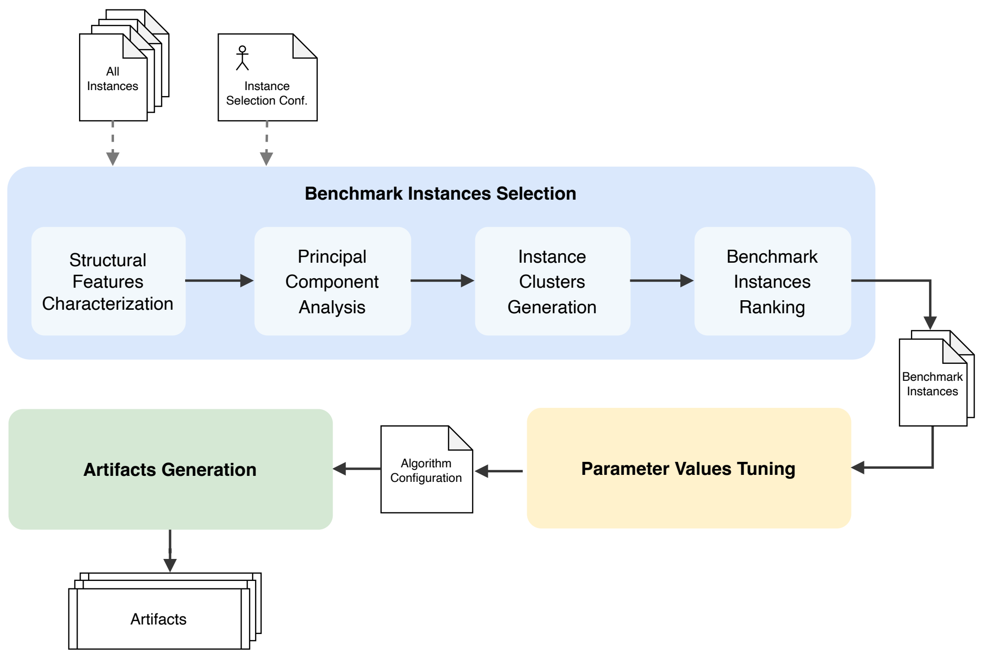

All Instances
Instance Selection Conf.
Benchmark Instances Selection
Structural Features Characterization
Principal Component Analysis
Instance Clusters Generation
Benchmark Instances Ranking
Benchmark Instances
Artifacts Generation
Algorithm Configuration
Parameter Values Tuning
Artifacts

Figure 1: Proposed methodology to favor reproducibility.

according to structural features, returning a number of representative instances which are selected as benchmark instances. In the second main step, benchmark instances are used to tune the parameter values of the algorithm, generating the configuration that should be used in the experimental experience. Finally, in the third main step, the algorithm configuration will be used to produce the corresponding artifacts.

The proposed methodology is in line with the one described in López-Ibáñez et al. (2021). As principal outcomes, it allows researchers to conduct studies about repeatability, reproducibility, replicability, and generalizability. For the sake of completeness, we include the definition of these concepts, as well as the definition of random and fixed factors in the context of performance assessment of stochastic algorithms:

- *Random factors* correspond to those elements whose value may belong to a certain distribution, but only a random subset of values will be evaluated in the experiment. The typical random factors are the random seeds.
- *Fixed factors*, as in evolutionary computation, include target instances, parameter values, time limits, and so on. Converting fixed factors into random factor allows the generalization of conclusions from an experiment.
- *Repeatability* consists in repeating the proposed experiment, obtaining the very same results. This kind of study requires the availability of the same instances, artifacts, and run environment from the original study. As will be later described in more detail, the software framework used is able to generate a Docker container with the full run environment, including the same random factors, in order to guarantee that the results are identical after each execution.

- *Reproducibility* analyzes the influence of random factors over the results of an experiment. In the proposed methodology, an independent artifact is provided which automates the benchmark instance selection. In addition, an external tool is used to select the parameter values of the algorithm. Finally, the algorithm and source code are provided as artifacts. Therefore, a researcher may vary the random seeds in any of these elements, including the benchmark selection process.

- *Replicability* compares the results from a new implementation of the artifacts using new random factors. In this case, since our methodology thoroughly describe the benchmark instances selection process and the algorithm configuration, and the source code is also available, any replication is possible.

- *Generalizability* comes after replicability studies also analyzing new fixed factors. Again, since all the steps of the methodology are detailed, a researcher may change the fixed factors and compare the obtained results. Notice that elements of the benchmark selection like the features of the instances are also fixed factors which can be changed.

Next, each one of the proposed steps of our methodology is thoroughly described, as well as the selected implementation for the target problem.

2.1 Benchmark instance selection

Computer scientists have mainly considered two different strategies to deal with instances, which are based on separating them into two different subsets. The first one divides the set of instances between "training set" (used to configure/tune the proposed algorithm) and "testing set" (used to compare the proposed algorithm with the state-of-the-art procedures), the intersection of these two sets being empty. This strategy is the one mainly adopted by the Machine Learning (ML) community. The second strategy, divides the set of instances also into two sets. On the one hand, the preliminary instance set or benchmark instances, which is devoted to tuning/configuring the algorithm (equivalent to the training set). However, the comparison of the proposed algorithm with the state-of-the-art procedures is conducted over the whole set of available instances. This strategy is the one followed by the heuristic optimization community. As aforementioned, we validate the proposed empirical methodology by proposing a metaheuristic method to solve a specific optimization problem. Therefore, we consider an approach commonly used in the heuristic optimization community.

As a departing hypothesis, we consider that the larger the studied set of instances, the more precise the expected outcomes of the algorithm. Therefore, we presume that researchers will provide as many instances as possible for the corresponding experiments. According to the number of instances, it is customary to choose a representative percentage of the total number of instances. The design of this benchmark instance set is critical, affecting directly to the performance of an algorithm. As stated in De Souza et al. (2021), a common mistake in the process of tuning an algorithm occurs when the instances selected in the preliminary set do not represent the whole set of available instances. Therefore, the configurations of the algorithm may be specialized on "known instances" (those included in the preliminary set) and perform poorly on "unknown instances" (those not included in the preliminary set).

The size of the benchmark instance set is usually selected arbitrarily, and it depends, among others, on the total number of instances of the problem, the time needed to perform the preliminary experiments, or the instances characteristics. Therefore, in this research, we do not go deeper into the decision of this value and let the researcher the decision of how many instances have to be selected in the benchmark set. On the contrary, we address the decision of defining the benchmark instance set, i.e., the choice of the instances that constitute the benchmark set.

We propose a semi-automatic benchmark selection process based on the four aforementioned principles: repeatability, reproducibility, replicability, and generalizability. To this end, we try to minimize the number of decisions based on the researcher's experience, by favoring those based on rational criteria. In particular, we adapt to the instance selection context some techniques commonly related to the statistical ML field. The idea of applying ML techniques in the metaheuristic domain to design efficient, effective, and robust algorithms has become increasingly popular during the last decades (Birattari et al., 2006; Olvera-López et al., 2010; Talbi, 2021). Among ML techniques, we propose the use of Principal Component Analysis (PCA) and *k*-means clustering algorithms to select the benchmark instances.

As seen in Figure 1, the benchmark selection process receives as input the whole set of available instances of the problem. Additionally, it is provided a user-defined configuration that includes the number of instances to be selected, and the parameters required for the initialization of the PCA and *k*-means algorithms (seed, confidence level, number of repetitions, etc.). Then, four steps are sequentially executed: Structural Features Characterization, Principal Component Analysis, Instance Clusters Generation, and Benchmark Instance Ranking. These steps are explained below.

Structural Features Characterization. In this first step, the instances are read and processed to obtain, for each one of them, a set of metrics and features. This step needs the intervention of the researcher, since it requires identifying and describing the features that are relevant for a given problem. Then, those features are extracted by the code implementation of the researcher. In many cases, understanding the problem domain helps to detect which features can be useful.

Principal Component Analysis. The previous step generates an output matrix of metrics and features, i.e., a multivariate data set. However, a high number of instance features hinders the analysis of the data for extracting conclusions. This issue specifically gets worse when features have different scales or measurement units. Occasionally, this is not an issue at all, depending on the target problem. Therefore, we propose to ensemble two well-known techniques to deal with it: standardization of the data and PCA.

Data standardization is a common requirement for ML algorithms and consists of transforming the data by removing the mean value of each feature and then scaling it by dividing each value by its standard deviation. Therefore, the application of data standardization results in a new set of values where the mean is zero and the standard deviation is one.

Once the data have been standardized, we perform the PCA. This technique is generally used to reduce the redundant information of a given set of variables. This procedure generates a new set of features, denoted as the principal components set, that retains much of the information from the original set, although the number of variables included has been significantly reduced (Jolliffe, 2005).

To determine the number of principal components, we use the explained variance ratio, i.e., the percentage of variance that is attributed to each one of the selected components. Generally, we would like to select a number of principal components that explain around the 90% of the data (Jolliffe and Cadima, 2016). We have implemented this process to be run automatically. However, the percentage of attributed variance has to be established by the researcher in the input configuration file of the procedure.

Instance Clusters Generation. The result of the second step is a reduced set of variables that represent the features of the instances. In this third step, we use the *k*-means algorithm to classify the instances in clusters, according to the principal components previously obtained. The *k*-means algorithm is one of the most widely used clustering methods. It aims to partition a set of observations into *k* clusters, where more similar elements are classified in the same cluster (MacQueen et al., 1967).

The number of clusters is usually specified by the researcher. However, a simple and popular solution to determine the optimal number of clusters is the elbow method. The elbow method consists of executing the *k*-means algorithm for a determined range of *k* values. Then, for each cluster obtained in each one of the *k* executions, the average of the distortion score (the sum of the squared distances from each point to its assigned cluster center) is computed. Finally, the optimal value of *k* is determined visually as the point where the slope of the curve has a significant change (like an elbow) when plotting the obtained scores. In order to automate the process of determining the number of clusters and to avoid possible mistakes by the researcher, we have implemented the method proposed by Satopaa et al. (2011), which detects the optimal number of clusters and, therefore, automating this process. Briefly, the authors proposed a formal definition for identifying a knee in discrete data sets based on the mathematical definition of curvature for continuous functions. We omit the details for the sake of brevity.

As a result of this third step, we obtain a classification of the instances into an appropriate set of clusters based on the initial structural features.

Benchmark Instances Ranking. The fourth and final step of the proposed methodology aims to determine the instances that will become part of the benchmark set. For this purpose, the *k* clusters are sorted in descending order according to their size, i.e., the number of instances. Then, they are sequentially traversed, selecting from each of them the instance that minimizes the distance to its centroid. This automatic process is repeated until the desired number of instances of the benchmark set is reached. It is worth mentioning that the selected instances are removed within its cluster, so no instances can be repeated in the benchmark set, and if a cluster is empty, it is skipped and the next non-empty cluster is visited.

Hence, the selected benchmark instances represent the diversity of the complete set of instances according to their structural features. Moreover, the selection is not made by the researcher, but given by these automatic procedures, supported by the described methods.

The proposed methodology in this section has been implemented using Jupyter Notebooks using Python. Section 5.1 presents the application of the proposed methodology in the context of the DRFLP. All artifacts and source code are publicly available, as will be detailed in Section 6.

2.2 Parameter values tuning

Generally, metaheuristic approaches require tuning several parameters. These parameters make the proposed algorithm robust, but also flexible, and have a great influence on its effectiveness and efficiency. The problem of finding the optimal parameters for an algorithm could be considered itself as a hard continuous optimization problem, where the objective is to optimize the performance of a given optimization algorithm over a set of preliminary instances.

Researchers have generally classified the tuning strategies in two types: offline parameter tuning and online parameter tuning. The first strategy consists of fixing the algorithm parameters before execution, while in the second strategy, the parameters are dynamically adapted during execution. This work is mainly focused on the first strategy, the offline parameter tuning.

Customarily, parameter tuning is done manually, and it includes a difficult and time-consuming test approach guided by the experience of the researcher. Particularly, manual tuning involves a preliminary exploration to find a suitable range of parameters. Then, tests are executed to evaluate only a single parameter at a time, while the others parameters are set to a rational value. The best value of the parameter tested is set for the next experiment. These systematic tests are performed until all algorithm parameters have been tuned (Crainic et al., 1993; Gendreau, 2003).

Although manual testing could be effective for adjusting a few parameters, it has many drawbacks when the algorithm to be optimized is complex. For instance, it has difficulties in adjusting algorithms in which there are cross effects between parameters, the design alternatives that are explored are limited by the number of experiments, the tuning process is irreproducible, or some configurations are discarded due to bad performance in a particular experiment, among others (Stützle and López-Ibáñez, 2019).

To mitigate the disadvantages of manual tuning, researches have proposed advanced automatic tuning procedures that do not require their intervention. These procedures, usually denoted as Automatic Algorithm Configuration (AAC) procedures, follow a common general scheme: first, they receive as input the parameters to configure of the algorithm, as well as their domain, range and restrictions; then, they generate a list of candidate configurations of the algorithm which are evaluated over a benchmark set of instances; finally, the best configurations are returned. Examples of these AAC procedures are ParamILS (Hutter et al., 2009), Sequential Model-based Algorithm Configuration (SMAC) (Hutter et al., 2011), and `irace` (López-Ibáñez et al., 2016).

In this work, we propose the use of `irace` for automatically tuning the algorithm proposals. `irace` is an iterated racing method based on the Friedman-Race (F-Race) and the iterated F-Race (I/F-Race). Particularly, the `irace` program requires a user-defined input configuration that defines the behavior of the algorithm to be tuned. The evaluation of the configurations is done through a racing procedure where each configuration is evaluated for each instance of the benchmark set. The `irace` software has been implemented in R and is freely available at http://iridia.ulb.ac.be/irace/.

Section 5.2 describes the use of `irace` on the DRFLP, as well as the obtained results.

2.3 Artifacts generation

There exist several technical solutions that allow the portability of software to different environments such as virtual machines and containers like Docker, and platforms like Open Science Foundation or Code Ocean, for instance (Clyburne-Sherin et al., 2019). These technical solutions include everything needed to replicate an experiment (code, system tools, system libraries, and settings, and independence of the underlying operating system). Nevertheless, once this validation step is done, the preservation of code and data is more useful in the long-term than the availability of a reproducible experimental environment. This is due to the rapid obsolesce of software/hardware. Even if the original study becomes non-reproducible due to the obsolescence of its original artifacts, studying their code and data could help future replication and generalization efforts.

According to the definitions from literature, an artifact is a digital object that was part of the research process, either created or being an external tool used in this process

(ACM, 2021). In addition, instances, result files, etc. are also considered artifacts.

Therefore, taking into account the previously mentioned four different types of reproducible studies according to López-Ibáñez et al. (2021), our methodology proposes the generation of a number of mandatory artifacts, which are listed below:

- **Problem instances and results files**. Not only the instance files to be read by the algorithm but also, if possible, explanatory text files regarding technical questions like, for instance, file format. In addition, plain text files (or spreadsheet files) with the obtained results.
- **Source code of the proposed methods**. These files include proposed software for instance processing (if needed), proposed algorithms and methods to generate the final results with the metrics detailed in the work.
- **Executable artifacts**. Binary or executable versions on a portable environment for all the elements described in the previous item.
- **Analysis artifacts**. Generated diagrams, R scripts or any tool used to generate them, with the raw results used to generate any related figure or summary table.

Giving access to these artifacts allows the development of any kind of reproducibility study. As it will be later described, some of these artifacts were automatically generated by the framework we have developed.

3 Double Row Facility Location Problem

Despite the main proposal of the work is to provide a practical methodology for reproducibility, we also describe how this methodology has been applied to a particular problem from the family of Facility Layout Problems (FLP), where we have also improved the results from the state of the art.

The FLP family comprises a number of different optimization problems that are devoted to determine the optimal position of a given set of facilities in a particular layout. This family of problems has many interesting applications in Flexible Manufacturing Systems (FMS), where facilities have to be positioned on a particular layout where, usually, an automated machine carries material among facilities (Satheesh Kumar et al., 2008). Hence, the objective of the optimization problem is to minimize a cost function which is related to the material handling cost between facilities.

3.1 State of the art

Several layouts can be found in the related literature. For instance, the simplest layout, where all facilities are consecutively located, is known as the Single Row Facility Location Problem (SRFLP) (Simmons, 1969; Rubio-Sánchez et al., 2016). Some other layouts are the multiple row layout (Herrán et al., 2021), the circular layout (Hungerländer et al., 2020), and the T-row layout (Dahlbeck, 2021), among others.

In this paper, we have selected one of the most common layouts in the literature, which is the use of two rows to arrange the facilities allowing free space separation between them, known as the Double Row Facility Location Problem (DRFLP). Notice that the variant of this problem where no empty space is allowed between facilities is known as the Space-Free Double Row Facility Location Problem (SF-DRFLP) or Corridor Allocation Problem (CAP) (Amaral, 2012).

The DRFLP was firstly tackled from a heuristic point of view in Chung and Tanchoco (2010). In this work, five heuristics were compared with a branch & cut algorithm solving a MIP model. One of the heuristics was finally used to generate an initial solution for the exact algorithm, and its behavior was compared in a set of 16 instances from 6 to 36 facilities. The DRFLP was also studied in Amaral (2013), proposing a new mathematical model smaller than the one proposed by Chung and Tanchoco. The experimental results, however, do not compare the same instances as in the previous paper. Besides, some years later, the mathematical model was enhanced in Secchin and Amaral (2019) improving the results of Amaral (2013) in instances up to 16 facilities. In Chae and Regan (2020), the authors propose another mathematical model obtaining competitive results for instances up to 16 facilities. Following his work on this problem, Amaral recently published a work improving the mathematical model once again (Amaral, 2021) reporting better results than Chae and Regan (2020) and, finally, a different work proposing a two-phase algorithm which merges a heuristic method with a linear programming routine (Amaral, 2020). This is the paper with the highest number of instances and, in addition, with the largest sizes of the state of the art.

As seen, many works have proposed exact algorithms for this problem, and some of them have included heuristic methods. However, reproducibility has not been promoted in any of them. One of the reasons for this situation could be that there are no algorithmic parameters to tune due to the nature of the proposed methods. Besides, the small size of the instances studied in the state of the art prevents the design of an automated selection of instances. In addition, regarding the heuristic methods, there is no publicly available code or artifacts, and no effort has been made to group the studied instances in the most recent papers.

3.2 Description of the problem

The Double Row Facility Location Problem (DRFLP) is an $\mathcal{NP}$-hard optimization problem that consists in finding an optimal arrangement of rectangular facilities in two rows.

Let F be the set of facilities (with $n = |F|$), where each facility $i \in F$ has an associated length l_i, and w_{ij} is the amount of flow between two facilities $i, j \in F$. Then, an instance I of the DRFLP is defined by a triplet $I = (F, L, W)$, where L is a vector (of size n) containing all the facility lengths and W is a squared matrix (of size $n \times n$) of pairwise flows. Let us illustrate with an example all the relevant information of an instance of the DRFLP. To this end, we consider the instance *S9* from Amaral (2020) with $n = 9$ facilities, represented with capital letters (from $\mathtt{F}_1$ to $\mathtt{F}_9$). Specifically, Table 1 shows the cost between each pair of facilities (W), and the length of each facility (L). Figure 2 shows a feasible solution π for the instance introduced in Table 1, where x_i is the abscissa of the center of facility i when it is assigned to any row of the layout. As can be seen in Figure 2, all the facilities in F must be placed at some position x_i but with no overlapping between consecutive facilities in each row of the layout.

A solution to this optimization problem is to find an optimal mapping π that assigns the set of facilities F to the corresponding layout with two rows, together with the exact horizontal location of the center of each facility i in the layout (abscissa x_i). Specifically, given a facility $i \in F$, $\pi^k(p) = i$ indicates that i is located at position p of the permutation corresponding to row k in the layout π. Hence, the complete layout is defined by two permutations of facilities, one for each row, $\pi = \{\pi^1, \pi^2\}$. It trivially holds that the number of facilities of a feasible solution is $|\pi| = |\pi^1| + |\pi^2| = n$. For the example shown in Figure 2, $\pi^1 = \{\mathtt{F}_2, \mathtt{F}_3, \mathtt{F}_4, \mathtt{F}_8\}$ and $\pi^2 = \{\mathtt{F}_7, \mathtt{F}_5, \mathtt{F}_6, \mathtt{F}_9, \mathtt{F}_1\}$. In addition to the permutation, the location of centers x_i must be provided since free space can be found in a solution.

Table 1: Instance data for *S9* from Amaral (2020).

W	F$_1$	F$_2$	F$_3$	F$_4$	F$_5$	F$_6$	F$_7$	F$_8$	F$_9$
F$_1$	0	0	2	8	7	4	0	1	6
F$_2$	0	0	8	0	2	7	4	4	6
F$_3$	2	8	0	2	7	8	0	2	6
F$_4$	8	0	2	0	5	0	8	8	6
F$_5$	7	2	7	5	0	5	4	7	6
F$_6$	4	7	8	0	5	0	8	2	6
F$_7$	0	4	0	8	4	8	0	4	6
F$_8$	1	4	2	8	7	2	4	0	6
F$_9$	6	6	6	6	6	6	6	6	0
L	2	8	9	7	3	4	6	8	9

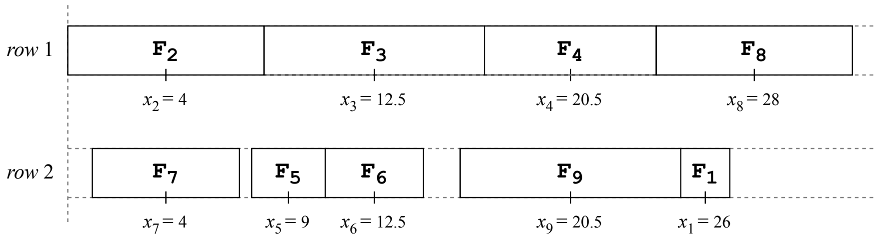


Figure 2: Feasible solution of the DRFLP for the instance introduced in Table 1.

Then, given an instance I and a solution defined by a permutation π together with the centers x_i of each facility $i \in F$, the objective function of the DRFLP, denoted as $\mathcal{F}(I, \pi)$, is computed as the total weighted sum of the center-to-center distances between each pair of facilities. This cost, usually known as the material handling cost, can be formulated in mathematical terms as it is shown in Equation (1). According to the data introduced above, the solution shown in Figure 2 has a cost $\mathcal{F}(I, \pi) = 1607$.

$$\mathcal{F}(I, \pi) = \sum_{1 \leq i < j \leq n} w_{ij} \cdot |x_i - x_j| \tag{1}$$

The optimization problem then consists in finding the permutation π together with the centers x_i of each facility $i \in F$ that minimizes the aforementioned objective function subjected to the following constraints to avoid the overlapping between consecutive facilities in each row:

$$x_{\pi^k(p)} \geq x_{\pi^k(p-1)} + \frac{l_{\pi^k(p-1)}}{2} + \frac{l_{\pi^k(p)}}{2} \tag{2}$$

3.3 Combinatorial approach

As stated before, a solution to the DRFLP is defined by two permutations of facilities, one for each row, $\pi = \{\pi^1, \pi^2\}$, together with the exact horizontal location of the center of each facility i in the layout (abscissa x_i). The location of the centers is required since an optimal solution of the DRFLP could have a free space between consecutive facilities located in the same row, and the permutation is not able to store this information without the centers.

If the length of each facility is an integer number greater or equal than 1, it can be demonstrated that the allowed free spaces between consecutive facilities in the optimal solution (if they exist) is a multiple of 0.5 units[1]. Hence, a modification in the solution representation can be made in order to represent the complete layout (free spaces included) in the permutation.

In particular, we incorporate "dummy facilities" of length 0.5, whose weight between them and any other facility (including other dummy facilities) is equal to zero. Hence, adding a number of these "dummy facilities" to the original instance I, makes it possible to find the optimal solution of an instance that could need free spaces between facilities by only dealing with permutations. This way, centers x_i are not considered as decision variables in this process.

Figure 2 shows a feasible solution $\pi = \{\pi^1, \pi^2\}$ of the DRFLP for the instance introduced in Table 1 considering 6 "dummy facilities" of length 0.5 denoted as d_i to model free spaces between facilities. The permutations for these solutions are the following, $\pi^1 = \{\mathtt{F}_2, \mathtt{F}_3, \mathtt{F}_4, \mathtt{F}_8\}$ and $\pi^2 = \{d_1, d_2, \mathtt{F}_7, d_3, \mathtt{F}_5, \mathtt{F}_6, d_4, d_5, d_6, \mathtt{F}_9, \mathtt{F}_1\}$.

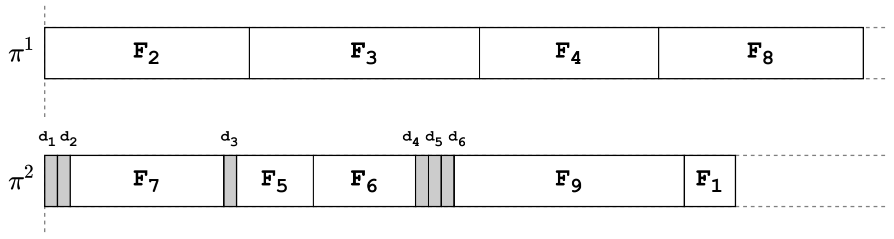


Figure 3: Solution representation of example depicted in Figure 2 including "dummy facilities".

Then, once the set of "dummy facilities" D is included in the original instance $I = I \cup D$, the optimization problem then consists in finding the solution $\pi^\star$ that minimizes the aforementioned objective function without considering the center of each facility as a decision variable. More formally:

$$\pi^\star \leftarrow \arg\min_{\pi \in \Pi} \mathcal{F}(I, \pi) \tag{3}$$

where Π represents the set of all feasible layouts for the DRFLP.

4 Algorithmic approach for the DRFLP

Previous works on the DRFLP have designed heuristic methods which provided good-quality initial solutions for exact methods, as in Amaral (2020). On the contrary, our proposal is based on applying a metaheuristic method to tackle this problem. In particular, an Iterated Greedy algorithm is designed where the constructive and improvement phases are based on the Greedy Randomized Adaptive Search Procedure (GRASP), followed by a destructive method. In addition, the algorithm is embedded in a multi-start loop which is responsible for introducing spaces between facilities by means of "dummy facilities" (see Section 3.3).

[1]The minimum difference between two facilities is 1 unit. If they are centered, a space of 0.5 units is generated at each side of the smallest of both, being the smallest gap possible.

Next, the algorithmic proposals are described, ending the section with the complete structure of the final algorithm.

4.1 Iterated Greedy algorithm

We have selected the Iterated Greedy (IG) algorithm (Jacobs and Brusco, 1995) as the main component of our algorithmic proposal. It is a very fast and simple method based on consecutive partial destruction and re-construction of a candidate solution. Due to its simple design and efficient performance, it has been applied to many different problems (Ruiz and Stützle, 2007; Lozano et al., 2011; Ruiz et al., 2019; Quintana et al., 2021).

Algorithm 1 shows the pseudocode of our IG proposal, which receives the following input parameters: I, which is the target instance; α_1 and α_2, which balance the random bias of the constructive methods; i_{max}, which corresponds to the number of iterations of the IG inner loop; and β, which determines the damage ratio in the destructive method. As seen in step 1 of the pseudocode, an initial solution π is generated from scratch. This initial solution is improved by a local search procedure and stored as the current best solution $\pi^\star$ in step 2. Then, the inner loop begins with the partial destruction of solution $\pi^\star$ in step 4. The resulting solution, π', is re-constructed in step 5 and then improved again by a local search in step 6, obtaining a new solution π'''. Finally, steps 7 to 8 represent the acceptance criterion of our IG proposal, which is a simplified version of the one used in the original IG proposal from Ruiz and Stützle (2007). In particular, a re-constructed solution is accepted for the next iteration only if its cost function value is better than the current best solution $\pi^\star$. Therefore, $\pi^\star$ is updated accordingly and returned by the algorithm in step 9 after the corresponding number of iterations i_{max}.

Algorithm 1: Iterated Greedy($I, \alpha_1, \alpha_2, i_{max}, \beta$)

```
1 $\pi \leftarrow$ InitialConstructiveMethod($\alpha_1$)
2 $\pi^\star \leftarrow$ LocalSearch($\pi$)
3 for $i = 1$ to $i_{max}$ do
4     $\pi' \leftarrow$ DestructiveMethod($\pi^\star, \beta$)
5     $\pi'' \leftarrow$ ReconstructionMethod($\pi', \alpha_2$)
6     $\pi''' \leftarrow$ LocalSearch($\pi''$)
7     if $\mathcal{F}(I, \pi''') < \mathcal{F}(I, \pi^\star)$ then
8         $\pi^\star \leftarrow \pi'''$
9 return $\pi^\star$
```

4.2 Constructive methods

In order to either create an initial solution or reconstruct an incoming solution, we have designed two different approaches based on the Greedy Randomized Adaptive Search Procedure (GRASP). GRASP was proposed by Feo and Resende (1989) as a new method where greediness and randomness are balanced in the construction of a solution.

Algorithm 2 shows the pseudocode of our first constructive proposal, called *C1*. This method builds a solution by iteratively adding new facilities to the partial layout, which is initially empty. The method starts by adding a randomly selected facility i at the beginning of the first row of an empty layout $\pi^1(1)$ in steps 1 and 2. Then, after creating a candidate list CL with the rest of facilities in step 3, the method enters a loop until all the facilities in CL are allocated in the final layout π, returned in step 13. To this purpose, an extended candidate list ECL is created in step 5. Notice that this list contains all the combinations (i, k, p) of facilities $i \in CL$ and the positions $1 \leq p \leq |\pi^k| + 1$ in each row $1 \leq k \leq 2$ of the partial solution π where these facilities can be inserted. Then, a greedy function g is defined for the selection of a combination $(i, k, p) \in ECL$. Specifically, g evaluates the contribution to the objective function when a facility $i \in CL$ is inserted at position p of row k, resulting in a new partial solution π', $g(I, \pi, i, k, p) = \mathcal{F}(I, \pi') - \mathcal{F}(I, \pi)$. Once the minimum and maximum values of this function are computed in steps 6 and 7, respectively, a threshold value th is calculated in step 8 to generate the reduced candidate list RCL from ECL in 9. This list is populated with all the elements in ECL whose greedy function g is below the previously calculated threshold value th, which depends on α. Finally, the loop ends by randomly selecting one element (i, k, p) from RCL in step 10, and inserting the selected facility i at the corresponding position (k, p) of the layout π in step 11. Notice that the insertion of a facility i in position (k, p) increases the center (abscissa) of all the facilities in positions from p to $|\pi^k|$ row k in l_i units. Once the facility i is included in the partial layout π, it is removed from CL in step 12 and a new iteration stars computing the new ECL at step 5.

```
Algorithm 2: C1 $(I, \alpha)$
1  $i \leftarrow$ SelectRandom$(F)$
2  $\pi^1(1) \leftarrow i$
3  $CL \leftarrow F \setminus \{i\}$
4  while $|CL| > 0$ do
5      $ECL \leftarrow \{(i,k,p) : \ i \in CL, \ 1 \leq k \leq 2, \ 1 \leq p \leq |\pi^k| + 1\}$
6      $g_{min} = \min_{(i,k,p) \in ECL} g(I, \pi, i, k, p)$
7      $g_{max} = \max_{(i,k,p) \in ECL} g(I, \pi, i, k, p)$
8      $th \leftarrow g_{min} + \alpha \cdot (g_{max} - g_{min})$
9      $RCL \leftarrow \{(i,k,p) \in ECL : g(I, \pi, i, k, p) \leq th\}$
10     $(i, k, p) \leftarrow$ SelectRandom$(RCL)$
11     $\pi \leftarrow$ InsertFacility$(\pi, i, k, p)$
12     $CL \leftarrow CL \setminus \{i\}$
13 return $\pi$
```

As seen, this proposal follows the customary GRASP-constructive approach, which performs a greedy selection followed by a random choice of a candidate. An alternative to this approach is to exchange the random and greedy stages to perform random selections and, then, to select the best facility in terms of the greedy function. Therefore, we propose a new constructive procedure called $C1'$ where the main changes are in the while-loop. Specifically, steps 6 to 10 in Algorithm 2 are substituted by first taking at random a percentage α of elements from ECL (i.e., the number of selected elements is $\alpha \cdot |ECL|$) and, finally, selecting the best one of these elements according to the corresponding greedy function. We have omitted the pseudocode description of this method for the sake of space, since the source code will be provided.

We propose a third constructive method, called $C2$, where a solution is constructed by iteratively merging facilities to produce partial solutions until all the facilities are included in the layout. Algorithm 3 shows the pseudocode of our second constructive proposal $C2$. The algorithm starts by creating the candidate list CL with n partial solutions (each containing one of the n facilities $i \in F$) in step 1. Then it enters a while-loop (steps 2 to 10) which reduces the number of partial solutions in CL in one unit at each iteration by merging a pair of solutions $\pi_a, \pi_b \in CL$. This loop ends when all the partial solutions are merged into a single solution ($|CL| = 1$), which is returned in step 11. In this case, the extended candidate list ECL is generated with all the combinations of pairs of partial solutions $\pi_a, \pi_b \in CL$ and moves $m \in M$, (π_a, π_b, m) to merge them into a new resulting partial solution. Figure 4 shows the set M of available moves for merging two solutions. The greedy function g' here computes the increase in the objective function value of a partial solution π' resulting from applying move m to combine π_a and π_b: $g(I, \pi_a, \pi_b, m) = \mathcal{F}(I, \pi') - \mathcal{F}(I, \pi_a) - \mathcal{F}(I, \pi_b)$. After computing the minimum and maximum values of this function in steps 4 and 5, a threshold value th is calculated in 6 to generate the reduced candidate list RCL from ECL in 3. Finally, the loop ends by randomly selecting one element (π_a, π_b, m) from RCL in step 8, and performing the selected move m to merge π_a and π_b into a new partial solution π in step 9. Then, solutions π_a and π_b are removed from CL, and the partial solution π is added to CL in step 10.

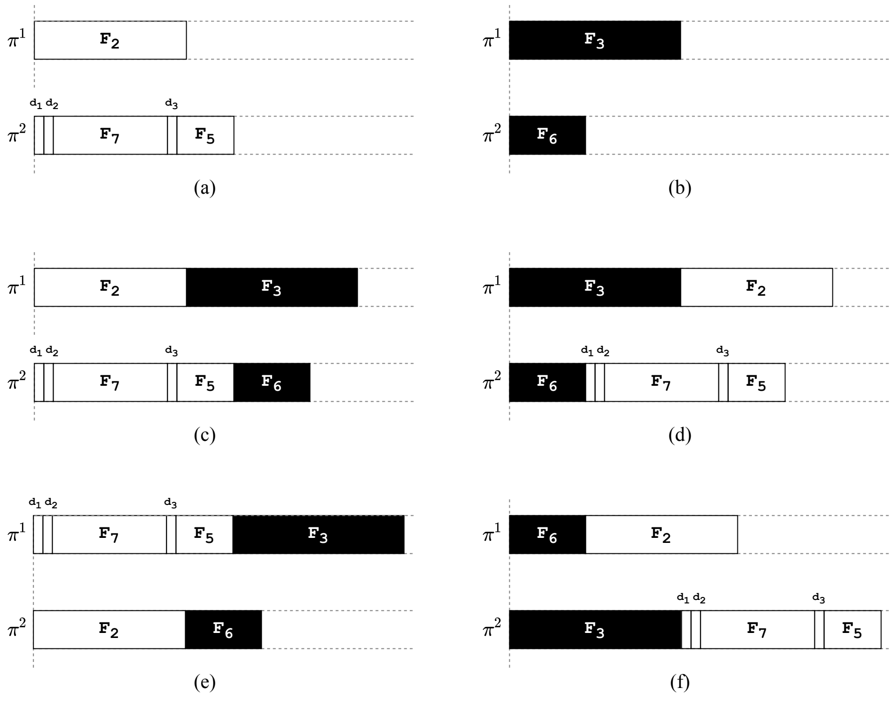


Figure 4: Available moves for constructive procedure $C2$ considering (a) and (b) as partial solutions: (c) is the right-hand concatenation; (d) is the left-hand concatenation; (e) is the swapped right-hand concatenation where facilities of the left-hand partial solution changed the assigned row; and (f) is the swapped left-hand concatenation.

In addition to these GRASP-based constructive strategies, we propose a baseline method, called $C0$, devoted to construct a completely random solution. This method

Algorithm 3: C2 (I, α)

```
1  $CL \leftarrow$ GeneratePartialSolutions$(F)$
2  while $|CL| > 1$ do
3      $ECL \leftarrow \{(\pi_a, \pi_b, m) : \ \pi_a, \pi_b \in CL, \ m \in M\}$
4      $g'_{min} = \min_{(\pi_a,\pi_b,m) \in ECL} g'(I, \pi_a, \pi_b, m)$
5      $g'_{max} = \max_{(\pi_a,\pi_b,m) \in ECL} g'(I, \pi_a, \pi_b, m)$
6      $th \leftarrow g'_{min} + \alpha \cdot (g'_{max} - g'_{min})$
7      $RCL \leftarrow \{(\pi_a, \pi_b, m) \in ECL : g'(I, \pi_a, \pi_b, m) \leq th\}$
8      $(\pi_a, \pi_b, m) \leftarrow$ SelectRandom$(RCL)$
9      $\pi \leftarrow$ MergePartialSolutions$(\pi_a, \pi_b, m)$
10     $CL \leftarrow CL \setminus \{\pi_a\} \setminus \{\pi_b\} \cup \{\pi\}$
11 return $\pi \in CL$
```

is equivalent to executing either $C1$ or $C2$ with $\alpha = 1$. The goal of including this method is to evaluate whether the use of information about the problem (e.g., $C1$ and $C2$) improves the outcomes or not.

4.3 Local search

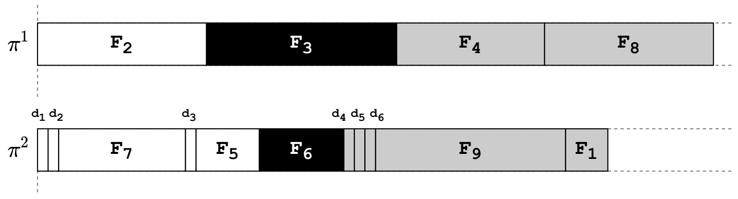
As it is shown in Algorithm 1, our IG implementation includes a local search procedure to improve the solution generated in the construction phase. Specifically, we have used the *best improvement* strategy to explore the neighborhoods generated by *insert* moves. Moreover, we have also implemented the efficient computation of the cost associated to the neighbor solutions based on *swap* moves. See Herrán et al. (2021) for a detailed explanation of *insert* moves and the efficient exploration strategy.

4.4 Destructive methods

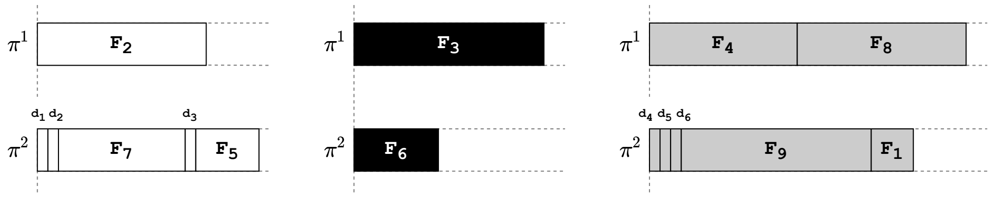
In this work we have implemented two different destructive methods depending on how the initial solution is constructed. The first method, called *D1*, randomly removes a number of facilities (controlled by the parameter β) of the current solution π.

The second destructive method, called *D2*, tries to split the current solution into several partial solutions. First, it randomly selects a facility $i \in \pi^1$, and then, it randomly selects a facility $j \in \pi^2$ that is located in a position "overlapped" by i. For example, considering the solution depicted in Figure 5 (a), if the method selects the facility $i = \mathtt{F}_3$, a facility j is randomly selected among $\mathtt{F}_5$, $\mathtt{F}_6$, $\mathtt{F}_9$ or any of the three "dummy facilities" between $\mathtt{F}_6$ and $\mathtt{F}_9$ since they all are overlapped by F_3. Assuming that the method selects $j = \mathtt{F}_6$, the original solution is split into the three partial solutions depicted in Figure 5(b): those facilities located on one side of the selected facilities (left-hand side of the figure); both the selected facilities (center of the figure); and those facilities located on the other side of the selected facilities (right-hand side of the figure). Notice that the facilities of these outgoing solutions are aligned to the origin abscissa. Hence, this method can be applied again to the two partial solutions different to the selected facilities.

Since we will provide the source code of our proposals, we have omitted the pseudocode of these methods for the sake of space.



(a)



(b)

Figure 5: Destructive procedure for the solution shown in Figure 3.

4.5 Final proposal

In this paper we propose a multi-start Iterated Greedy algorithm (MS-IG) which we have modified in order to increasingly include “dummy facilities” into the corresponding instance I while the current solution can be improved. Algorithm 4 shows the pseudocode of our final proposal. The algorithm begins by setting to 0 the number of “dummy facilities” of the instance to be solved (d) and the number of “dummy facilities” of the instance associated to the best current solution found ($d^{\star}$) in steps 1 and 2, respectively. It also calculates in step 3 the threshold value th used for the termination criteria of the while loop. The threshold is defined as the maximum increment of fake facilities since last improvement, or, in other words, how many fake facilities can be added to a solution without improving before stopping. This value depends on the current instance, and it will be one of the parameters of the algorithm to tune. Next, the IG method described in Algorithm 1 is applied to solve the original instance I (without including any “dummy facility”) in step 4. Then, the algorithm enters a while-loop which applies t_{max} times at each iteration the IG method, but using a new instance I' containing additional “dummy facilities” (step 9). This new instance I' is updated from I in step 8 according to the new number of “dummy facilities” calculated in step 7. The strategy for increasing fake facilities can be easily customized. In this case, we propose two different implementations: linearly increment fake facilities after each iteration by a constant or exponentially increasing them using a function or sequence such as Fibonacci. Again, this will be another parameter to tune. After solving the new problem in step 9, steps 10 to 13 update the best current solution found π, instance I and number of “dummy facilities” $d^{\star}$ if some improvement on the objective function cost value is achieved in step 10. The loop in steps 5 to 13 iterates while the difference between the number of “dummy facilities” of the instance I' solved at each iteration and the number of “dummy facilities” of the instance I associated to the best current solution is below the threshold value th calculated in step 3.

Once again, since the source code is available in Zenodo, we omit the pseudocode

Algorithm 4: Multi-Start Iterated Greedy($I, \alpha, i_{max}, \beta, t_{max}$)

```
1  $d \leftarrow 0$
2  $d^\star \leftarrow d$
3  $th \leftarrow$ GetThreshold($I$)
4  $\pi \leftarrow$ IteratedGreedy($I, \alpha, i_{max}, \beta$)
5  while $(d - d^\star) < th$ do
6      for $t = 0$ to $t_{max}$ do
7          $d \leftarrow$ IncreaseDummyFacilities($d$)
8          $I' \leftarrow$ IncludeDummyFacilities($I, d$)
9          $\pi' \leftarrow$ IteratedGreedy($I', \alpha, i_{max}, \beta$)
10         if $\mathcal{F}(I', \pi') < \mathcal{F}(I, \pi)$ then
11             $\pi \leftarrow \pi'$
12             $I \leftarrow I'$
13             $d^\star \leftarrow d$
14 return $\pi$
```

of `GetThreshold`, `IncreaseDummyFacilities` and `IncludeDummyFacilities` methods.

5 Results

Once the proposed methodology is described, the target problem defined and the algorithmic proposal detailed, this section is devoted to show the particular application of the methodology to the DRFLP. Similarly, we will also show the results obtained by our approach in relation to the state of the art on the target problem. Finally, since we propose a new set of larger instances, we have been forced to re-implement the algorithmic proposal from the state of the art because no code or binaries were provided by the authors. We then describe this replicability study as well as the obtained results.

All the algorithmic components have been implemented using Java 17, under our developing framework MORK[2]. In addition, all experiments have been executed on the same virtual machine, provided with 32 cores, 16 GB of RAM and Ubuntu Server 20.04 as the operating system. We have also used Gurobi 9.1.2 as an exact solver in the replication of the state-of-the-art algorithms.

5.1 Instances under study

In the case of the DRFLP, state-of-the-art algorithms have been tested on a set of 38 synthetic instances. These instances have been proposed in several papers related to Facility Layout Problems (Amaral, 2006, 2019; Anjos and Vannelli, 2008; Secchin and Amaral, 2019; Simmons, 1969). Additionally, we have extended the set of instances by introducing 15 new larger instances also used in the FLP problem family (Maadi et al., 2017).

According to the topology and structure of the DRFLP, instances can be understood as a weighted and undirected graph. The vertices correspond to facilities, which have a given size corresponding to the width of the facility. The weight of the edges will correspond to the pairwise material handling cost between facilities. If this value is zero, no edge will connect two vertices. Therefore, we apply some common graph

[2]`https://doi.org/10.5281/zenodo.6241726`

	Avg. Vert.	Avg. Edges	Avg. Width	Avg. Weight	Density
State-of-the-art set (38)	20.47	162.90	7.82	9.23	0.66
New proposed set (15)	62.00	1290.93	24.79	2.53	0.67

Table 2: Comparison between the state-of-the-art set of instances and the new proposed set of instances.

metrics for both instance sets. Particularly, we show in Table 2 the average number of vertices (Avg. Vert.), the average number of edges (Avg. Edges), the average facility width (Avg. Width), the average edge weight (Avg. Weight), and density. The selected metrics are able to describe the main features of the instances, with no redundant information. Both instance sets and the analysis can be publicly found as artifacts as specified in Section 6.

Given the complete set of 53 instances, the proposed Benchmark Instance Selection process was executed in order to determine the benchmark set. First, the general properties of the procedure are configured. Specifically, the number of instances of the benchmark set is established as 15% of the total sample size, i.e., 8 instances. According to the number of components of the PCA, we select the minimum number of components whose explained variance is greater than 90%. Regarding the *k*-means algorithm configuration, *k*-means++ is chosen for the selection of the initial values of the algorithm (further details of the *k*-means++ initialization algorithm can be found in Arthur and Vassilvitskii (2006)). Besides, the maximum number of iterations is set to 1000 and the maximum number of clusters is set to 15. Once the input parameters for the process have been specified, the four steps detailed in Section 2.1 are performed.

The Benchmark Instance Selection process starts with the structural features characterization. Considering the instances as graphs, we propose the following structural properties: number of vertices, number of edges, maximum/minimum/average/standard deviation of the degree of the vertices, maximum/minimum/average/standard deviation of the size of the vertices, maximum/minimum/average/standard deviation of the weight of the edges and density of the graph. Hence, these 15 features were obtained for each one of the 53 instances.

In the second step, we scale the data through standardization. Then, we execute the PCA to obtain the cumulative explained variance for a number of principal components in the range [0,14]. The result of the PCA is presented in Figure 6. Particularly, it can be observed that the 4 variables are enough to explain more than the 90% of the data. In addition, we represent the individual explained variance of each component with green bars.

Given the obtained 4 principal components, we execute multiple times the *k*-mean algorithm with *k* in the range $[2-14]$. Then, for each value of *k* the distortion score is reported and plotted in order to determine the optimal number of clusters using the elbow method. The resultant chart is depicted in Figure 7. As it can be observed, the "elbow" occurs when $k=5$. Therefore, the instances will be classified into 5 clusters.

Finally, the benchmark instance selection process concludes by selecting the most representative instances of each cluster, i.e., the closest instances to the centroid of each cluster, following a descending order. The number of instances selected of each cluster depends on the cardinality of it. Specifically, the selected 8 instances are: 40-02, 40-04, 40-06, A60_03, A70_03, sko56_04, 14a and Am13a. Moreover, in Table 3, we report the same graph metrics as in Table 2 for the selected instances and the cluster they belong to. As it can be easily observed, there is a clear distinction between instances of each

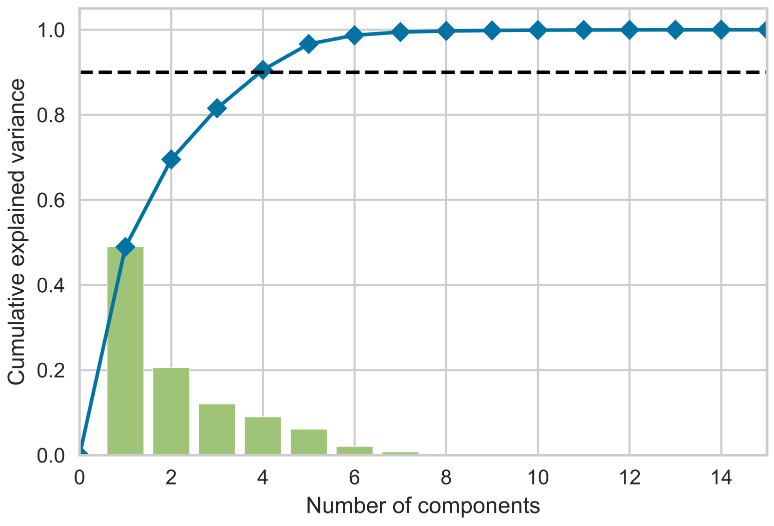


Figure 6: cumulative explained variance by the number of components computed using PCA.

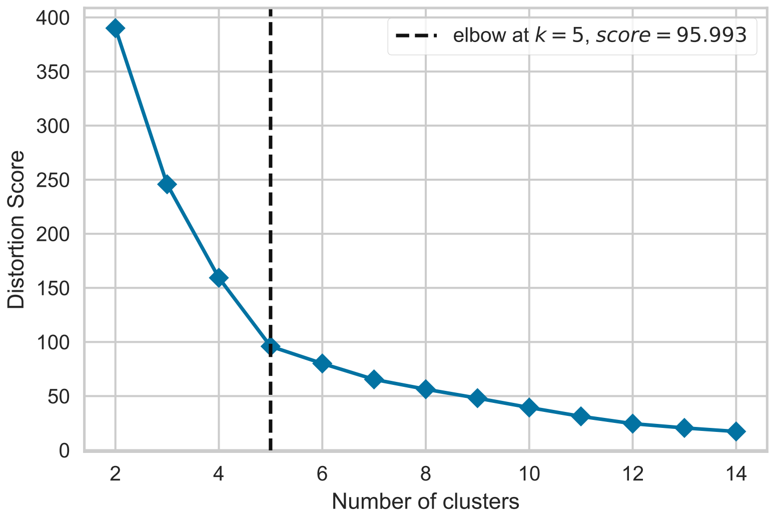


Figure 7: Finding the optimal number of clusters using the Elbow method for *k*-means clustering algorithm.

Cluster	Instance	Vert.	Edges	Avg. Width	Avg. Weight	Density
0	40-02	40	484	10.10	12.17	0.62
	40-04	40	391	9.13	11.51	0.50
1	40-06	40	694	11.25	50.24	0.89
2	A60_03	60	1076	31.05	1.61	0.61
	A70_03	70	1669	38.26	1.52	0.69
3	sko56_04	56	1061	5.52	4.02	0.69
4	14a	14	69	6.57	4.25	0.76
	Am13a	13	56	6.85	4.21	0.72

Table 3: Selected instances of each cluster after the benchmark instance selection process.

cluster. For instance, selected instances from clusters 0 and 1, which share the same number of vertices, differ in other features like number of edges and average weight. Similarly, instance from cluster 3 presents an average width very different to instances from cluster 2, which are closer in terms of number of edges. A similar analysis can be performed for all the different combinations of clusters.

5.2 Automatic algorithm configuration

Following the guidelines of the proposed methodology, the parameters of the algorithm were adjusted with an automated method. In particular, the algorithm has been tuned using `irace` (López-Ibáñez et al., 2016).

Table 4 summarizes the parameters given to `irace`. In particular, "constructive" decides which constructive implementation to use in the initial constructive method (step 1 in Algorithm 1). Their possible values are: "random", corresponding to the random generator of solutions denoted as $C0$ in Section 4.2, "graspgr" and "grasprg", which correspond to the greedy-random and random-greedy methods $C1$ and $C1'$ (Algorithm 2); and "tetris", which corresponds to $C2$ (Algorithm 3). Parameter "alpha1" represents the α_1 parameter for the initial constructive method. The reconstruction method (step 5 in Algorithm 1) is firstly configured by using the "reconstructive" parameter, which specifies the reconstructive implementation to use. Notice that it receives the same set of values as the initial constructive method. Parameter "alpha2" defines the α_2 parameter for the reconstruction phase and "destratio" defines the percentage of the solution to destroy during the destruction phase, denoted as β in Algorithm 1. Notice that there is no parameter for the destructive method (step 4 in Algorithm 1) since $D2$ destruction is required in case any variant of $C2$ is selected, since they both are based on combinations of partial solutions (see Section 4.4). On the contrary, $D1$ is selected if any variant of $C1$ is chosen by `irace`. The following parameters correspond to the multi-start configuration. Parameter "stop" defines the strategy used to determine the threshold to stop (step 3 in Algorithm 4): by fraction or by constant values. Hence, depending on the selected value, a "fractionv"fraction for a given instance, or "constantv" constant value must be selected. The "increment" parameter defines how the fake facilities are incremented (step 7 in Algorithm 4): linearly with "linearinc" value or following the Fibonacci function with the "fiboinc" value. Parameter "linearratio" refines the value of the linear increment. Finally, since `irace` is able to choose the number of iterations for both the multi-start and the Iterated Greedy components, `irace` will try to maximize both as they will never worsen the final result, getting an increment of the total computing time. Instead of providing a maximum number of iterations for both components to `irace`, the parameter "iterationsratio" is defined as the proportion of iterations spent in the outer for-loop of the MS-IG (t_max in Algorithm 4) compared with the inner for loop of the IG method (i_max in Algorithm 1) such as $outerIterations * IGIterations = 1000000$. This way, `irace` is able to balance the time spent diversifying with different solutions or intensifying working on the same solution.

This configuration is described in the "parameters.txt" file inside the `irace` folder in the repository specified in Section 6.

Table 4: Parameter definitions for `irace` tuning.

Name	Type	Restriction
constructive	category	$\{graspgr, grasprg, tetris, random\}$
alpha1	real	$[0, 1]$
reconstructive	category	$\{graspgr, grasprg, tetris, random\}$
alpha2	real	$[0, 1]$
destratio	real	$[0, 1]$
stop	category	{fraction, constant}
fractionv	real	$[0, 0.5]$
constantv	integer	$[0, 40]$
increment	category	{linearinc, fiboinc}
linearratio	integer	$[1, 20]$
iterationsratio	ordinal	$\{v : 1000000 \mod v = 0\}$

Using a maximum budget of 1000 experiment, and limiting the total execution time of any algorithm to 60 seconds, the best algorithm configurations found by `irace` are summarized in Table 5.

Table 5: Top three configurations found by `irace`. NA means a configuration value is not applicable to the current algorithm. "destratio" omitted as it is not applicable to any of the chosen algorithm configurations.

#	constr	alpha1	reconst	alpha2	iterationsratio	stop	fractionv	constantv	increment	linearratio
1	graspgr	0.73	tetris	0.30	200	fraction	0.21	NA	linearinc	11
2	graspgr	0.78	tetris	0.05	160	fraction	0.27	NA	linearinc	9
3	grasprg	0.57	tetris	0.36	400	fraction	0.36	NA	linearinc	6

Elite configurations have several properties in common: first, the constructive chosen for all configurations is $C1$. Both variants (greedy random and random greedy) have high values for "alpha1", which means that the constructive method favors diverse solutions, accepting suboptimal moves during the construction phase. The best reconstruction method is $C2$, and during reconstruction, the "alpha2" parameter is more restrictive. The preferred stop or threshold method is using a fraction of the instance size, and best results are achieved by incrementing the number of fake facilities linearly each iteration. Finally, the "iterationsratio" is balanced, or in other words, the number of iterations will be distributed uniformly between the iterated greedy and the multistart loops.

The best algorithm configuration found, represented as $\#1$, is selected for the comparison with the state of the art.

5.3 Comparison with the state of the art

Once the best algorithm configuration has been determined by `irace`, its performance is assessed over the complete set of 53 problem instances.

Firstly, our Multi-Start Iterated Greedy (MS-IG) proposal is compared with the four heuristics proposed in Amaral (2020) (H1, H2, H3 and H4) under the same set of 38 instances tackled in that paper. Table 6 illustrates this comparison, showing the averaged objective function value (Avg. o.f.), execution time in seconds, percentage deviation to the best value found (% Dev.), and number of best values found (# Best) for each algorithm, highlighting in bold font the best results. As seen, our MS-IG proposal obtains all the best values (38 of 38), spending short computation times. Detailed tables with objective function values and execution times per instance are available in Annex A in tables 8 and 9 respectively. Notice that a fair comparison of execution times cannot be made, since the previous work does not report the characteristics of the computer where the experiments were run.

Table 6: Summary comparison between the state-of-the-art algorithm and our proposal, using the set of 38 instances from Amaral (2020).

	H1	H2	H3	H4	MS-IG
Avg. o.f	101143.9	101137.1	101139.2	101137.1	**101117.0**
Time (s)	8575.2	7456.2	9335.7	9246.0	**27.2**
% Dev.	0.01%	0.01%	0.01%	0.01%	**0.00%**
# Best	31	31	26	27	**38**

Due to the fact that the previous work has not made any executable, artifact or source code available, we have implemented their proposal following the descriptions in Amaral (2020). To ease future comparisons, we have made it available on the same code repository as our proposal, including an explanation of the correspondence between each coded instruction with Amaral's algorithm.

In this small replicability study, we have also assessed the validity of our implementation of the state-of-the-art method in order to verify that the comparison is fair. In this way, the results reported in Amaral (2020) are compared to our implementation of the previous methods using the original set of instances, obtaining a maximum, minimum and average deviations over the results of 2.41%, -0.02%, and 0.32%, respectively. Despite these differences, which could be attributed to fine-grain implementation details, we believe that our implementation is correct. Complete tables comparing both the objective function values and execution times between each heuristic pair for each instance can be found in Annex A, in tables 10 and 11 respectively. Notice that our implementations of Amaral's methods are labeled as H1', H2', H3' and H4'.

Note that, probably due to hardware differences and programming language used in the previous work computer and ours, the execution times are different. In particular, as seen in Table 11, our implementation of the state-of-the-art heuristics is up to 10 times faster.

Once the accuracy of the previous heuristics implementations has been established, a new experiment is executed with the set of 18 larger instances. The previous heuristics are limited to one hour execution time, while the MS-IG is limited to a maximum time of 600 seconds. Both algorithms are executed in the same machine, using the same runtime libraries. A summary of the results is presented in Table 7, where the MS-IG proposal obtains all the best values, with an average execution time one order of magnitude lower. Notice that the previous heuristics reach the time limit in all runs. As with the previous experiment, detailed tables comparing both the objective function values and execution times can be found in Annex A, in tables 12 and 13 respectively. Hence, the MS-IG proposal is able to obtain the best results in all the 53 problem instances, spending short computation times in relation to the state-of-the-art.

Table 7: Summary comparison between the state-of-the-art algorithm and our proposal, using the new proposed set of instances.

	H1′	H2′	H3′	H4′	MS-IG
Avg. f.o	501776.6	502013.0	501494.5	501260.7	**499954.4**
Time (s)	3600.8	3600.8	3600.8	3600.6	**429.4**
% Dev	0.37%	0.47%	0.28%	0.25%	**0.00%**
# Best	0	0	0	0	**15**

Finally, we statistically compare the results of our MS-IG proposal in relation to the previous heuristic approaches. In particular, a performance analysis comparison of multiple algorithms over multiple instances simultaneously was performed under the Bayesian approach described in Calvo et al. (2018, 2019). This kind of analysis is able to produce a ranking of algorithms based on a probability distribution created after the results. Therefore, it calculates the expected probability of each algorithm to obtain better results than the others, denoted as probability of winning, giving credible intervals to this probability. Figure 8 shows the credible intervals (5% and 95% quartiles) and the expected probability of winning for all the methods in this comparison. In order to perform this analysis, the results from H1 to H4 were used for the 35 instances from the state of the art, while the results from our state-of-the-art implementation H1′ to H4′ were used for the 18 largest instances. All of them are represented by the H1 to H4 labels in the plot.

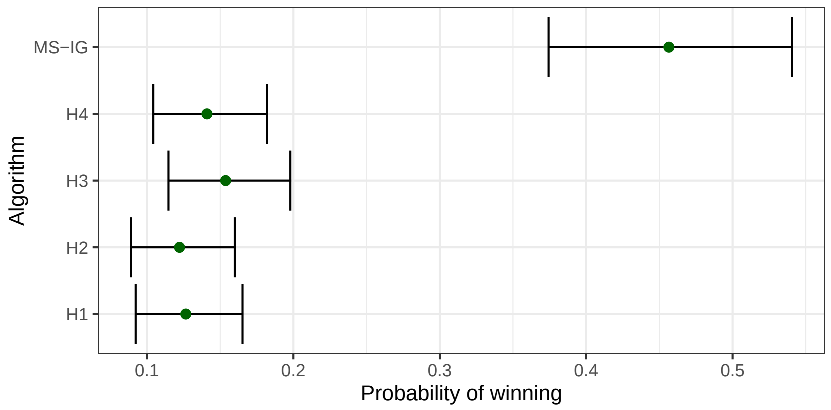


Figure 8: Credible intervals for the cost result obtained by all the heuristics approaches from Amaral (2020), and the proposed MS-IG.

As it can be seen in the figure, our MS-IG approach reaches higher probability values than the state-of-the-art methods. Besides, the credible intervals of all the state-of-the-art variants overlap, which means that their behavior is comparable. On the other hand, MS-IG approach reaches the highest probability of obtaining the best solution on average, 0.457, in relation to the other methods (< 0.15 on average). Moreover, the overall performance of this algorithm is clearly distinguished, since the credible intervals are not overlapped. Hence, the superiority of the proposed methods is not only supported by the results, but also for this Bayesian analysis.

6 Provided Artifacts and Reproducibility Discussion

All artifacts related to this paper are publicly available at GitHub[3] and archived in Zenodo[4], which in turn is indexed in OpenDOAR in order to ensure compliance with the FAIR principles (Wilkinson et al., 2016).

Following the recommendations from our methodology (see Section 2.3), the available artifacts are:

Problem instances and result files

- All instances used, both from the small and the new proposed set.
- Excel files with both the raw results data and the processed and summarized versions as used in this work.
- CSV and Excel files for the Bayesian analysis with the objective function results for all the instances and algorithms.

Source code of the proposed methods

- Python Jupyter notebook used to load, analyze and generate the benchmark instance set. The notebook can be easily adapted to work with instances from other problems.
- Java source code for both our proposal and our implementation of the state-of-the-art algorithms.
- Documentation detailing how to execute and reproduce the results, including extending the algorithms or using a new set of instances.

Executable artifacts

- The Python Jupyter notebook used for benchmark instance selection.
- Small Docker scripts that can build, run and deploy containers that easily run the proposal in any platform.

Analysis artifacts

- The R script used to generate the Bayesian analysis shown in Figure 8.
- Generated diagrams and tables.

Reproducibility Discussion

The proposed methodology as well as the provided artifacts ensure the reproducibility of our work on the DRFLP. Repeatability studies may be developed since all the executable artifacts and instances are publicly available. Reproducibility is possible since the source code for all the elements are available, allowing the researcher to modify any

[3]`https://github.com/GRAFO-URJC/DRFLP`
[4]`https://doi.org/10.5281/zenodo.6212684`

of the random factors. Replicability can be performed based not only in the provided source code, but also in the pseudo-code descriptions of all the methods. Finally, generalizability is also possible since all the steps of the study in the DRFLP are detailed, and the source code is also provided. Therefore, any fixed factors could be modified to compare with the proposed work.

7 Conclusions and Future Work

The aim of this paper is two-fold. On the one hand, we propose a practical methodology to reproduce the work on an optimization problem solved by means of metaheuristics. On the other hand, we have improved the state of the art on the Double-Row Facility Location Problem (DRFLP) applying the proposed methodology.

As seen in the paper, not only a set of precise steps have been described on the methodology, i.e., benchmark instances selection, parameter values tuning and artifact generation, but we have also provided actual artifacts that can be used by researchers in any kind of reproducibility studies. In particular, a Python notebook implementing all the methods described in the benchmark instances selection is available. Besides, our optimization framework MORK is connected to `irace` to automatically tune algorithmic parameters. In addition, the proposal is able to generate Docker containers with source code as well as including the same experimental environment of this work. Note that the applicability of these steps is not limited to using a specific software. Hence, a researcher may follow either the description in the paper or the source code to develop his/her own implementation, tune the algorithmic parameters with a different external software, and produce his/her own corresponding artifacts.

Regarding the DRFLP, we propose the application of a Multi-Start Iterated Greedy (MS-IG) algorithm, where four different constructive methods and two destructive methods have been proposed. Besides, we follow a model of the problem which introduces "fake facilities" as spaces between the actual facilities using two different strategies implemented in the MS-IG external loop. As stated before, we have followed our semi-automated methodology to select the instances and tune the algorithm in order to assure reproducibility studies.

In the experimental results, we have collected 38 instances from previous papers, and we also propose 15 new larger instances for this problem. In order to compare our MS-IG proposal in the larger instances, we have replicated the state-of-the-art method, obtaining similar results in the previous instances. This way, we have also shown the difficulties of these tasks when no artifacts are provided. As described in the experimental experience, our MS-IG proposal is able to obtain all the 53 best solutions spending a fraction of the execution time compared with the state of the art. Finally, a statistical analysis is conducted following a Bayesian approach in order to assess the performance of the different algorithms on the tackled set of instances.

Currently, we are studying different related lines regarding reproducibility, such as automatic characterization of instances. Furthermore, we pursue the creation of adimensional metrics to more precisely evaluate the performance of different implementations of algorithms over a problem. As in the case of this work, we will not only describe our findings, but we also will apply them on actual optimization problems.

Acknowledgments

This work has been partially supported by the Spanish Ministerio de Ciencia e Innovación (MCIN/AEI/10.13039/501100011033/FEDER, UE) under grant ref. PID2021-126605NB-I00; Spanish Ministerio de Ciencia, Innovación y Universidades

(MCIU/AEI/FEDER, UE) under grant refs. PGC2018-095322-B-C22 and FPU19/04098; and Comunidad de Madrid y Fondos Estructurales de la Unión Europea with grant ref. P2018/TCS-4566.

References

ACM (2021). Artifact review and badging, version 2.0.

Amaral, A. R. (2006). On the exact solution of a facility layout problem. *European Journal of operational research*, 173(2):508–518.

Amaral, A. R. (2012). The corridor allocation problem. *Computers & Operations Research*, 39(12):3325–3330.

Amaral, A. R. (2013). Optimal solutions for the double row layout problem. *Optimization Letters*, 7(2):407–413.

Amaral, A. R. (2019). A mixed-integer programming formulation for the double row layout of machines in manufacturing systems. *International Journal of Production Research*, 57(1):34–47.

Amaral, A. R. (2020). A heuristic approach for the double row layout problem. *Annals of Operations Research*, pages 1–36.

Amaral, A. R. (2021). A mixed-integer programming formulation of the double row layout problem based on a linear extension of a partial order. *Optimization Letters*, 15(4):1407–1423.

Anjos, M. F. and Vannelli, A. (2008). Computing globally optimal solutions for single-row layout problems using semidefinite programming and cutting planes. *INFORMS Journal on Computing*, 20(4):611–617.

Arnold, B., Bowler, L., Gibson, S., Herterich, P., Higman, R., Krystalli, A., Morley, A., O'Reilly, M., Whitaker, K., et al. (2019). The turing way: a handbook for reproducible data science. *Zenodo*.

Arthur, D. and Vassilvitskii, S. (2006). k-means++: The advantages of careful seeding. Technical report, Stanford.

Bäck, T., Fogel, D. B., and Michalewicz, Z. (1997). Handbook of evolutionary computation. *Release*, 97(1):B1.

Baker, M. (2016). 1,500 scientists lift the lid on reproducibility. *Nature*, 533(7604).

Bartz-Beielstein, T., Doerr, C., Berg, D. v. d., Bossek, J., Chandrasekaran, S., Eftimov, T., Fischbach, A., Kerschke, P., La Cava, W., Lopez-Ibanez, M., et al. (2020). Benchmarking in optimization: Best practice and open issues. *arXiv preprint arXiv:2007.03488*.

Birattari, M., Zlochin, M., and Dorigo, M. (2006). Towards a theory of practice in metaheuristics design: A machine learning perspective. *RAIRO-Theoretical Informatics and Applications*, 40(2):353–369.

Calvo, B., Ceberio, J., and Lozano, J. A. (2018). Bayesian inference for algorithm ranking analysis. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '18, pages 324–325, New York, NY, USA. ACM.

Calvo, B., Shir, O. M., Ceberio, J., Doerr, C., Wang, H., Bäck, T., and Lozano, J. A. (2019). Bayesian performance analysis for black-box optimization benchmarking. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '19, pages 1789–1797, New York, NY, USA. ACM.

Chae, J. and Regan, A. C. (2020). A mixed integer programming model for a double row layout problem. *Computers & Industrial Engineering*, 140:106244.

Chung, J. and Tanchoco, J. (2010). The double row layout problem. *International Journal of Production Research*, 48(3):709–727.

Clyburne-Sherin, A., Fei, X., and Green, S. A. (2019). Computational reproducibility via containers in psychology. *Meta-psychology*, 3.

Crainic, T. G., Gendreau, M., Soriano, P., and Toulouse, M. (1993). A tabu search procedure for multicommodity location/allocation with balancing requirements. *Annals of Operations research*, 41(4):359–383.

Dahlbeck, M. (2021). A mixed-integer linear programming approach for the t-row and the multi-bay facility layout problem. *European Journal of Operational Research*.

De Souza, M., Ritt, M., López-Ibáñez, M., and Pérez Cáceres, L. (2021). Acviz: A tool for the visual analysis of the configuration of algorithms with irace. *Operations Research Perspectives*, 8:100186.

Eiben, A. E. and Jelasity, M. (2002). A critical note on experimental research methodology in ec. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, volume 1, pages 582–587. IEEE.

Feo, T. and Resende, M. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71.

Gendreau, M. (2003). *An Introduction to Tabu Search*, pages 37–54. Springer US, Boston, MA.

Gent, I., Grant, S., Macintyre, E., Prosser, P., Shaw, P., Smith, B., and Walsh, T. (1997). How not to do it. Technical report, 97.27. School of Computer Studies, University of Leeds.

Glover, F. W. and Kochenberger, G. A. (2006). *Handbook of metaheuristics*, volume 57. Springer Science & Business Media.

Herrán, A., Colmenar, J. M., and Duarte, A. (2021). An efficient variable neighborhood search for the space-free multi-row facility layout problem. *European Journal of Operational Research*.

Hungerländer, P., Maier, K., Pachatz, V., and Truden, C. (2020). Exact and heuristic approaches for a new circular layout problem. *SN Applied Sciences*, 2(6):1–22.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer.

Hutter, F., Hoos, H. H., Leyton-Brown, K., and Stützle, T. (2009). Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306.

Jacobs, L. W. and Brusco, M. J. (1995). Note: A local-search heuristic for large set-covering problems. *Naval Research Logistics (NRL)*, 42(7):1129–1140.

Johnson, D. S. (2002). Experimental analysis of algorithms. *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges: Papers Related to the DIMACS Challenge on Dictionaries and Priority Queues (1995-1996) and the DIMACS Challenge on Near Neighbor Searches (1998-1999)*, 59:215.

Jolliffe, I. (2005). Principal component analysis. *Encyclopedia of statistics in behavioral science*.

Jolliffe, I. T. and Cadima, J. (2016). Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202.

Kendall, G., Bai, R., Błazewicz, J., De Causmaecker, P., Gendreau, M., John, R., Li, J., McCollum, B., Pesch, E., Qu, R., et al. (2016). Good laboratory practice for optimization research. *Journal of the Operational Research Society*, 67(4):676–689.

López-Ibáñez, M., Branke, J., and Paquete, L. (2021). Reproducibility in evolutionary computation. *ACM Transactions on Evolutionary Learning and Optimization*, 1(4):1–21.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., and Birattari, M. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

Lozano, M., Molina, D., and García-Martínez, C. (2011). Iterated greedy for the maximum diversity problem. *European Journal of Operational Research*, 214(1):31–38.

Maadi, M., Javidnia, M., and Jamshidi, R. (2017). Two strategies based on meta-heuristic algorithms for parallel row ordering problem (prop). *Iranian Journal of Management Studies*, 10(2):467–498.

MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1,14, pages 281–297. Oakland, CA, USA.

Moscato, P. and Norman, M. G. (1992). A memetic approach for the traveling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems. *Parallel computing and transputer applications*, 1:177–186.

Olvera-López, J. A., Carrasco-Ochoa, J. A., Martínez-Trinidad, J. F., and Kittler, J. (2010). A review of instance selection methods. *Artificial Intelligence Review*, 34(2):133–143.

Pineau, J., Vincent-Lamarre, P., Sinha, K., Larivière, V., Beygelzimer, A., d'Alché Buc, F., Fox, E., and Larochelle, H. (2021). Improving reproducibility in machine learning research: a report from the neurips 2019 reproducibility program. *Journal of Machine Learning Research*, 22.

Plesser, H. E. (2018). Reproducibility vs. replicability: a brief history of a confused terminology. *Frontiers in neuroinformatics*, 11:76.

Quintana, J. D., Martin-Santamaria, R., Sanchez-Oro, J., and Duarte, A. (2021). Solving the regenerator location problem with an iterated greedy approach. *Applied Soft Computing*, 111:107659.

Ralph, P., Ali, N. b., Baltes, S., Bianculli, D., Diaz, J., Dittrich, Y., Ernst, N., Felderer, M., Feldt, R., Filieri, A., et al. (2020). Empirical standards for software engineering research. *arXiv preprint arXiv:2010.03525*.

Rubio-Sánchez, M., Gallego, M., Gortázar, F., and Duarte, A. (2016). Grasp with path relinking for the single row facility layout problem. *Knowledge-Based Systems*, 106:1–13.

Ruiz, R., Pan, Q.-K., and Naderi, B. (2019). Iterated greedy methods for the distributed permutation flowshop scheduling problem. *Omega*, 83:213–222.

Ruiz, R. and Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European journal of operational research*, 177(3):2033–2049.

Satheesh Kumar, R., Asokan, P., Kumanan, S., and Varma, B. (2008). Scatter search algorithm for single row layout problem in fms. *Advances in Production Engineering & Management*, 3(4):193–204.

Satopaa, V., Albrecht, J., Irwin, D., and Raghavan, B. (2011). Finding a" kneedle" in a haystack: Detecting knee points in system behavior. In *2011 31st international conference on distributed computing systems workshops*, pages 166–171. IEEE.

Secchin, L. D. and Amaral, A. R. S. (2019). An improved mixed-integer programming model for the double row layout of facilities. *Optimization Letters*, 13(1):193–199.

Simmons, D. M. (1969). One-dimensional space allocation: an ordering algorithm. *Operations Research*, 17(5):812–826.

Stodden, V., Leisch, F., and Peng, R. D. (2014). *Implementing reproducible research*, volume 546. CRC Press Boca Raton, FL.

Stützle, T. and López-Ibáñez, M. (2019). Automated design of metaheuristic algorithms. In *Handbook of metaheuristics*, pages 541–579. Springer.

Talbi, E.-G. (2021). Machine learning into metaheuristics: A survey and taxonomy. *ACM Computing Surveys (CSUR)*, 54(6):1–32.

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., Bourne, P. E., et al. (2016). The fair guiding principles for scientific data management and stewardship. *Scientific data*, 3(1):1–9.

A Detailed results of the experiments

In order to facilitate future comparisons, the complete set of results for every algorithm and instance combination is provided in this annex.

Tables 8 and 9 show, respectively, the objective function values and total execution time in seconds obtained by the methods described in Amaral (2020) (denoted as H1 to H4) and our MS-IG proposal on the set of 38 instances from Amaral (2020). The best results are highlighted in bold font. As can be seen, the MS-IG proposal reaches all the best values, spending only a fraction of the total computing time spent by the previous methods.

Table 8: Full comparison of objective function values between the state of the art and our MS-IG proposal, using the original set of 38 instances from Amaral (2020).

Instance	H1	H2	H3	H4	MS-IG
S9	**1179**	**1179**	**1179**	**1179**	**1179**
S9H	**2293**	**2293**	**2293**	**2293**	**2293**
S10	**1351**	**1351**	**1351**	**1351**	**1351**
11a	5559.5	5559.5	**5559**	5560.2	**5559**
11b	**3655.5**	**3655.5**	**3655.5**	**3655.5**	**3655.5**
11c	**3832.5**	**3832.5**	**3832.5**	**3832.5**	**3832.5**
11d	**906.5**	**906.5**	**906.5**	**906.5**	**906.5**
11e	**578**	**578**	**578**	**578**	**578**
11f	**825.5**	**825.5**	825.7	**825.5**	**825.5**
S11	**3424.5**	**3424.5**	**3424.5**	**3424.5**	**3424.5**
12a	**1493**	**1493**	**1493**	**1493**	**1493**
12b	**1606.5**	**1606.5**	**1606.5**	**1606.5**	**1606.5**
12c	**2012.5**	**2012.5**	**2012.5**	**2012.5**	**2012.5**
12d	**1107**	**1107**	**1107**	**1107**	**1107**
12e	**1066**	**1066**	**1066**	**1066**	**1066**
12f	**997.5**	**997.5**	997.7	997.6	**997.5**
13a	**2456.5**	**2456.5**	**2456.5**	**2456.5**	**2456.5**
13b	**2864**	**2864**	**2864**	**2864**	**2864**
13c	**4136**	**4136**	**4136**	**4136**	**4136**
13d	**6164.5**	**6164.5**	**6164.5**	**6164.5**	**6164.5**
13e	**6502.5**	**6502.5**	**6502.5**	**6502.5**	**6502.5**
13f	**7699.5**	**7699.5**	**7699.5**	**7699.5**	**7699.5**
14a	**2904**	**2904**	**2904**	**2904**	**2904**
14b	**2736**	**2736**	**2736**	**2736**	**2736**
P15	**3195**	**3195**	3195.3	**3195**	**3195**
P17	**4655**	**4655**	**4655**	**4655**	**4655**
N30_01	**4115**	**4115**	**4115**	**4115**	**4115**
N30_02	**10771**	**10771**	10773.5	**10771**	**10771**
N30_03	**22692**	22697	**22692**	**22692**	**22692**
N30_04	**28390**	**28390**	28393.5	28393	**28390**
N30_05	57400	**57393.5**	57395.5	57410.5	**57393.5**
40-01	99525.5	99543	99537	99531.5	**99492.5**
40-02	301002	300973.5	300992.5	300976	**300961.5**
40-03	416271.5	416277	416264	416257	**416254.5**
40-04	**207510**	**207510**	207511	207528	**207510**
40-05	**193748**	**193748**	193778	193783	**193748**
40-06	1881366.5	1881351.5	**1881277**	1881281.5	**1881277**
40-07	545474.5	545239	545358.5	545271	**544640**

Table 10 shows the results obtained by our implementation of the state-of-the-art heuristic, denoted as H1′ to H4′, in relation with the methods from Amaral (2020) for the set of 38 previous instances. The differences were calculated as a percentage deviation over the objective function score (columns %H1 to %H4). Total execution times in seconds are provided in Table 11.

Finally, Table 12 shows the objective function value obtained by our implementation of the state-of-the-art methods (H1′ to H4′) and our MS-IG proposal on the set of

Table 9: Full comparison of execution times between the state of the art and our MS-IG proposal, using the original set of 35 instances from Amaral (2020).

Instance	H1	H2	H3	H4	MS-IG
S9	64.1	59.4	32.4	32.1	4.6
S9H	62.3	58.2	28.7	28.4	3.2
S10	95.5	90.6	50.2	49.5	3.0
11a	135.7	132.7	75.8	75.1	29.2
11b	139.7	137.9	71.5	71.1	3.6
11c	136.2	133.9	71.6	71.2	3.1
11d	136.3	134	71.7	71.1	3.9
11e	136.6	134.8	68.9	69.2	4.6
11f	136	134.1	74.3	73.9	4.4
S11	142.7	135	69.7	69.2	3.8
12a	199.5	196.4	111.8	110.7	4.6
12b	198.9	195.8	103.3	102.8	4.7
12c	200.1	196.7	103.6	101.3	4.4
12d	201.6	186	94.1	92.7	4.4
12e	198.4	191	104.7	102.9	4.8
12f	197	193.9	109.1	108.4	4.3
13a	278	272.3	175.5	173.4	4.8
13b	278.7	271.5	182.6	180.7	5.3
13c	268.8	262.4	167.5	166.9	5.0
13d	271.7	266.5	168.7	165.8	5.1
13e	274.7	267.9	141.3	140.3	4.5
13f	271.6	265.2	159.9	158.4	5.3
14a	383.3	370.9	238.1	228.8	5.6
14b	380.1	370.4	232	228.5	5.2
P15	515	498.6	349.6	345.9	7.6
P17	892.8	870.3	639	631.5	7.5
N30_01	20296.2	19829	11597.8	11140.4	14.2
N30_02	12842.2	11779.5	8889.6	8679.2	26.7
N30_03	10805.1	9656.6	9231.2	9059.4	26.5
N30_04	10726.2	9712.9	10154.6	10051.1	30.9
N30_05	9529.7	8479.5	8952.2	8935.5	51.4
40-01	36940.4	31511.7	37697.9	37413.9	77.4
40-02	36351.6	30784.1	47770.6	47502.9	94.3
40-03	36710.5	31291.3	53366.1	52988.8	97.1
40-04	37832.6	32492.3	43152.6	42688.8	76.9
40-05	36249	31121.1	32599.3	32004.4	191.5
40-06	35919.9	30579.8	49446.2	49318	75.2
40-07	35458.1	30069.9	38202.2	37915	123.4

Table 10: Comparison between the objective function values of the previous algorithm (H1 to H4) and our implementation (H1′ to H4′).

Instance	H1	H2	H3	H4	H1′	H2′	H3′	H4′	%H1	%H2	%H3	%H4
S9	1179	1179	1179	1179	1181.5	1181.5	1181.5	1181.5	0.21	0.21	0.21	0.21
S9H	2293	2293	2293	2293	2293	2293	2293	2293	0.00	0.00	0.00	0.00
S10	1351	1351	1351	1351	1367	1367	1370.5	1370.5	1.18	1.18	1.44	1.44
11a	5559.5	5559.5	5559	5560.2	5631.5	5631.5	5621.5	5628.5	1.30	1.30	1.12	1.23
11b	3655.5	3655.5	3655.5	3655.5	3655.5	3655.5	3655.5	3655.5	0.00	0.00	0.00	0.00
11c	3832.5	3832.5	3832.5	3832.5	3832.5	3832.5	3832.5	3832.5	0.00	0.00	0.00	0.00
11d	906.5	906.5	906.5	906.5	906.5	906.5	906.5	906.5	0.00	0.00	0.00	0.00
11e	578	578	578	578	583	583	583	583	0.87	0.87	0.87	0.87
11f	825.5	825.5	825.7	825.5	827	827	827	827	0.18	0.18	0.16	0.18
S11	3424.5	3424.5	3424.5	3424.5	3439.5	3439.5	3439.5	3439.5	0.44	0.44	0.44	0.44
12a	1493	1493	1493	1493	1529	1529	1529	1529	2.41	2.41	2.41	2.41
12b	1606.5	1606.5	1606.5	1606.5	1609.5	1609.5	1609.5	1609.5	0.19	0.19	0.19	0.19
12c	2012.5	2012.5	2012.5	2012.5	2035	2035	2035	2035	1.12	1.12	1.12	1.12
12d	1107	1107	1107	1107	1112.5	1114	1112.5	1114	0.50	0.63	0.50	0.63
12e	1066	1066	1066	1066	1066	1066	1066	1066	0.00	0.00	0.00	0.00
12f	997.5	997.5	997.7	997.6	997.5	998	997.5	997.5	0.00	0.05	-0.02	-0.01
13a	2456.5	2456.5	2456.5	2456.5	2457.5	2467.5	2457.5	2457.5	0.04	0.45	0.04	0.04
13b	2864	2864	2864	2864	2870	2870	2870	2870	0.21	0.21	0.21	0.21
13c	4136	4136	4136	4136	4149	4149	4152	4152	0.31	0.31	0.39	0.39
13d	6164.5	6164.5	6164.5	6164.5	6164.5	6164.5	6164.5	6164.5	0.00	0.00	0.00	0.00
13e	6502.5	6502.5	6502.5	6502.5	6511.5	6511.5	6511.5	6511.5	0.14	0.14	0.14	0.14
13f	7699.5	7699.5	7699.5	7699.5	7699.5	7699.5	7699.5	7704.5	0.00	0.00	0.00	0.06
14a	2904	2904	2904	2904	2920	2920	2920	2920	0.55	0.55	0.55	0.55
14b	2736	2736	2736	2736	2746.5	2746.5	2736.5	2746.5	0.38	0.38	0.02	0.38
P15	3195	3195	3195.3	3195	3195	3206	3195	3206	0.00	0.34	-0.01	0.34
P17	4655	4655	4655	4655	4666	4666	4666	4666	0.24	0.24	0.24	0.24
N30_01	4115	4115	4115	4115	4115	4115	4115	4115	0.00	0.00	0.00	0.00
N30_02	10771	10771	10773.5	10771	10793.5	10781.5	10789.5	10782	0.21	0.10	0.15	0.10
N30_03	22692	22697	22692	22692	22710	22706	22711.5	22717	0.08	0.04	0.09	0.11
N30_04	28390	28390	28393.5	28393	28427	28417	28417	28430.5	0.13	0.10	0.08	0.13
N30_05	57400	57393.5	57395.5	57410.5	57481	57470.5	57485.5	57485	0.14	0.13	0.16	0.13
40-01	99525.5	99543	99537	99531.5	99874.5	99788	99727.5	99788	0.35	0.25	0.19	0.26
40-02	301002	300973.5	300992.5	300976	301216.5	301231.5	301265.5	301089	0.07	0.09	0.09	0.04
40-03	416271.5	416277	416264	416257	416401	416432	416328.5	416413	0.03	0.04	0.02	0.04
40-04	207510	207510	207511	207528	207670	207696	207693	207696	0.08	0.09	0.09	0.08
40-05	193748	193748	193778	193783	194047	194052	193957	194052	0.15	0.16	0.09	0.14
40-06	1881366.5	1881351.5	1881277	1881281.5	1881668.5	1881605.5	1881881.5	1881511.5	0.02	0.01	0.03	0.01
40-07	545474.5	545239	545358.5	545271	547137	548758	546671.5	546133	0.30	0.65	0.24	0.16

Table 11: Comparison between the execution times of the previous algorithm (H1 to H4) and our implementation (H1′ to H4′).

Instance	H1	H2	H3	H4	H1′	H2′	H3′	H4′
S9	64.1	59.4	32.4	32.1	26.8	22.0	5.9	5.5
S9H	62.3	58.2	28.7	28.4	22.1	19.3	5.3	5.3
S10	95.5	90.6	50.2	49.5	45.0	30.6	8.0	8.9
11a	135.7	132.7	75.8	75.1	62.4	46.2	13.9	11.9
11b	139.7	137.9	71.5	71.1	57.9	44.0	11.2	11.1
11c	136.2	133.9	71.6	71.2	57.5	44.2	12.5	12.3
11d	136.3	134.0	71.7	71.1	58.9	43.0	11.0	10.8
11e	136.6	134.8	68.9	69.2	56.8	43.7	11.1	10.1
11f	136.0	134.1	74.3	73.9	50.7	42.1	10.5	11.4
S11	142.7	135.0	69.7	69.2	51.2	43.2	10.0	8.8
12a	199.5	196.4	111.8	110.7	84.5	67.3	15.7	14.5
12b	198.9	195.8	103.3	102.8	71.0	62.6	15.2	14.3
12c	200.1	196.7	103.6	101.3	79.5	65.3	16.3	15.1
12d	201.6	186.0	94.1	92.7	82.4	62.8	16.7	16.7
12e	198.4	191.0	104.7	102.9	81.3	64.3	16.6	17.6
12f	197.0	193.9	109.1	108.4	80.4	60.2	16.2	14.6
13a	278.0	272.3	175.5	173.4	121.9	94.8	24.2	22.3
13b	278.7	271.5	182.6	180.7	114.6	94.1	22.6	22.7
13c	268.8	262.4	167.5	166.9	106.8	90.3	22.1	20.5
13d	271.7	266.5	168.7	165.8	107.2	89.4	22.8	22.1
13e	274.7	267.9	141.3	140.3	99.4	87.4	21.6	22.0
13f	271.6	265.2	159.9	158.4	101.8	76.6	17.7	18.5
14a	383.3	370.9	238.1	228.8	140.0	109.4	30.5	31.5
14b	380.1	370.4	232.0	228.5	124.4	104.6	26.2	31.6
P15	515.0	498.6	349.6	345.9	157.4	131.5	38.2	39.8
P17	892.8	870.3	639.0	631.5	256.3	201.4	64.2	63.5
N30_01	20296.2	19829.0	11597.8	11140.4	3466.1	2822.9	705.8	746.2
N30_02	12842.2	11779.5	8889.6	8679.2	3600.0	3308.0	799.9	874.3
N30_03	10805.1	9656.6	9231.2	9059.4	3600.0	3569.5	882.9	901.9
N30_04	10726.2	9712.9	10154.6	10051.1	3600.0	3600.0	912.1	963.7
N30_05	9529.7	8479.5	8952.2	8935.5	3600.0	3600.0	936.3	988.0
40-01	36940.4	31511.7	37697.9	37413.9	3600.1	3600.0	3600.1	3600.0
40-02	36351.6	30784.1	47770.6	47502.9	3600.1	3600.1	3600.1	3600.1
40-03	36710.5	31291.3	53366.1	52988.8	3600.1	3600.1	3600.1	3600.1
40-04	37832.6	32492.3	43152.6	42688.8	3600.1	3600.0	3600.0	3600.0
40-05	36249.0	31121.1	32599.3	32004.4	3600.1	3600.1	3600.1	3600.0
40-06	35919.9	30579.8	49446.2	49318.0	3600.1	3600.1	3600.1	3600.1
40-07	35458.1	30069.9	38202.2	37915.0	3600.1	3600.1	3600.1	3600.0

15 new instances. Table 13 provides the total execution times in seconds. The time limit is set to 3600 seconds for the state-of-the-art implementation and to 600 seconds for our MS-IG proposal. Note that although the state-of-the-art implementation reaches the time limit in every instance, and the MS-IG reaches it in those instances of size 70, the best solution found in that time is reported.

Table 12: Comparison of objective function values between our implementation of the state of the art and our proposal, using the new proposed set of 15 instances.

Instance	H1′	H2′	H3′	H4′	MS-IG
sko56_01	32021	32009	31982	32009	**31972**
sko56_02	249541	249258	248439	248385	**248201.5**
sko56_03	85393	85544	85265	85304	**85166.5**
sko56_04	157084	156883	156826	156838	**156626.5**
sko56_05	296357.5	296567.5	296438.5	296365.5	**296168.5**
A60_01	739879	740637.5	739464	739597.5	**738869**
A60_02	421256	421291	421268	421291	**420890**
A60_03	325343.5	325129.5	324887.5	325129.5	**324201.5**
A60_04	200260	200231	200068	200231	**199116**
A60_05	161468	163123	160576	160167	**159578**
A70_01	767469	768145	766473	766741.5	**764416**
A70_02	724458.5	721885	721885	721885	**720706**
A70_03	763116	762165.5	764582.5	759912	**759405**
A70_04	484917.5	489218	488639.5	489218	**484328**
A70_05	2118085.5	2118108.5	2115623.5	2115836.5	**2109671.5**

Table 13: Comparison of execution times between our implementation of the state of the art and our proposal, using the new proposed set of 15 instances.

Instance	H1′	H2′	H3′	H4′	MS-IG
sko56_01	3600.3	3600.7	3600.2	3600.5	98.7
sko56_02	3600.6	3600.6	3600.2	3600.3	405.9
sko56_03	3600.3	3600.6	3600.2	3600.5	162.4
sko56_04	3600.3	3600.5	3600.2	3600.4	181.8
sko56_05	3600.9	3600.6	3600.1	3600.1	211.0
A60_01	3600.9	3600.5	3600.5	3600.2	486.4
A60_02	3600.3	3600.6	3600.3	3600.9	453.1
A60_03	3600.2	3600.4	3600.6	3600.6	339.2
A60_04	3600.7	3600.3	3600.2	3600.7	467.0
A60_05	3600.4	3600.2	3600.2	3600.1	609.5
A70_01	3601.0	3601.7	3602.5	3601.7	605.7
A70_02	3600.2	3601.4	3600.9	3600.5	606.3
A70_03	3601.4	3601.3	3602.7	3601.8	606.1
A70_04	3602.0	3600.4	3601.5	3600.3	605.9
A70_05	3601.8	3601.6	3601.7	3601.0	602.0

Chapter 9

On the automatic generation of metaheuristic algorithms for optimization problems

Title	On the automatic generation of metaheuristic algorithms for optimization problems
Authors	Raúl Martín-Santamaría, José Manuel Colmenar, Abraham Duarte, Manuel López-Ibáñez and Thomas Stützle
Publication date	Submitted for review in May 2023
Journal	European Journal of Operational Research
Publisher	Science Direct
ISBN/ISSN	0377-2217
Impact Factor	6.363 (2021)
Rank by Impact Factor	17/87 (Q1, Operations Research & Management Science)
DOI	Pending assignment

On the automatic generation of metaheuristic algorithms for combinatorial optimization problems

Raúl Martín-Santamaría[a], J. Manuel Colmenar[a], Abraham Duarte[a], Manuel López-Ibáñez[b], Thomas Stützle[c]

[a]*Department of Computer Science and Statistics, Universidad Rey Juan Carlos, Móstoles, Spain*
[b]*Alliance Manchester Business School, University of Manchester, Manchester, UK*
[c]*IRIDIA, Université Libre de Bruselles, Brussels, Belgium*

Abstract

Metaheuristic algorithms have become one of the preferred approaches for solving optimization problems. However, some issues have been highlighted in the literature, among others, reproducibility, fragmentation and lack of reusability are some of the main challenges to overcome when designing metaheuristics algorithms. To this end, we propose a new methodology for automatically generating metaheuristic approaches using both existing common components and user specific or custom components. Moreover, we will implement the proposed methodology in a new optimization tool, demonstrating the benefits of the methodology by outperforming earlier research in three distinct problems from completely different families: a facility layout problem, a vehicle routing problem and a clustering problem.

Keywords: metaheuristics, methodology, reproducibility, automatic configuration

1. Introduction

Stochastic algorithms, and specially metaheuristics, are one of the most successful methods for solving optimization problems. Their distinct performance, specially when short computing times are required in practical applications, makes them considerably popular (Hoos and Stützle, 2004).

Email addresses: `raul.martin@urjc.es` (Raúl Martín-Santamaría), `josemanuel.colmenar@urjc.es` (J. Manuel Colmenar), `abraham.duarte@urjc.es` (Abraham Duarte), `manuel.lopez-ibanez@manchester.ac.uk` (Manuel López-Ibáñez), `stuetzle@ulb.ac.be` (Thomas Stützle)

Metaheuristic approaches design is mostly guided by human expertise and intuition and a great deal of trial-and-error, where the same problem-independent algorithmic components are recombined in various ways together with other problem-specific components. However, in recent times, there is a lot of interest in automatically designing metaheuristics from a library of algorithmic components given training instances of a problem.

Hyper-heuristics have been mostly used to generate or select low-level heuristics within a given algorithmic structure. Yet, there is very little separation between the algorithmic framework that implements the design space and the optimizer that searches that design space and instantiates new algorithms (Grefenstette, 1986). More recent approaches for the automatic design of metaheuristics have followed a different path, by clearly separating the algorithmic design framework and the automatic configuration tool, which we will call AC from this point on. This separation allows employing powerful off-the-shelf automatic configuration methods, initially designed for parameter tuning (or hyper-parameter optimization in machine learning) as AC. Early examples of this approach are SATenstein (KhudaBukhsh et al., 2009, 2016) and AutoMOACO (López-Ibáñez and Stützle, 2012a).

While the parametric tuning of algorithms has been studied in depth, structural tuning has received less attention. In Stützle and López-Ibáñez (2019), the authors discuss the problem and metaheuristic specific automated design proposals, noting the lack of a general framework. Previous automated configuration proposals, such as KhudaBukhsh et al. (2016), are usually restricted to applying a limited set of metaheuristic methods to a single optimization problem family. In addition, the proposal from KhudaBukhsh et al. (2016) is a monolithic framework: there is a high-level algorithmic template with various parameters. This imposes a rather inflexible algorithmic structure. However, this inflexibility can be overcome by the use of grammars to define the design space (Mascia et al., 2014).

One of the most recent and relevant proposals, the `Emili` framework (Pagnozzi and Stützle, 2019), follows a more interesting approach. The authors of `Emili` propose splitting the algorithm components into high-level components, common for different problems, and low-level components, specific for a given problem, and used by the high-level components. This idea, proposed previously by Marmion et al. (2013), is improved by the `Emili` framework by providing the researcher with several high-level components. However, the user must manually modify the grammar defined by `Emili` and the interface with the AC to work, which is an error-prone process.

A similar limitation is shared by the ParadisEO framework. ParadisEO is an optimization framework written in C++ with a long track record (Cahon et al., 2004) that recently has incorporated an automating configuration module (Dréo et al., 2021). However, it shares a similar limitation with `Emili`: grammars still need to be manually defined. Moreover, the framework user needs to implement specific C++ code in order to transform each parameter to the corresponding parameter format of the AC tool, and to instantiate each component. This approach, while functional, requires a lot of user work.

To this end, we have not found an existing proposal that will automatically discover the algorithmic components, their relationships and dependencies given a source code, and automatically design a new algorithm. On the contrary, previous works require users to explicitly enumerate all the components, their parameters, and their relationships; a process that is tedious and error-prone. Moreover, since the user has to manually specify the design space in the format of the selected AC method, such as the case of the ParadisEO framework, the user is locked to a single AC tool, as parameter information needs to be provided using the tools' specific format.

In Swan et al. (2022), some key ideas that influence our research are presented. Specifically, the authors argue that automatic design frameworks should be "truly extensible algorithm templates that support reuse without modification", which is closely related to the well known SOLID design principles (Martin, 2011). In particular, the Liskov substitution principle (Martin, 1996a), also known as *design by contract*; and the *open-close principle* (Martin, 1996b), that states that components should be extendable rather than modifiable. Both principles will be key aspects of the proposed component-based design. The former, because by definition any algorithmic component will be able to be swapped by any other component that satisfies the same requirements. For example, all constructive methods can be used in place of each other.

The second principle, because components themselves cannot be modified. The application of the second principle may be counter intuitive at first, as not allowing users to modify existing framework components may give the impression that the framework is not flexible. This is a trap in which many existing frameworks have fallen. By allowing users to modify reference components, they may no longer match the same base requirements, and therefore components stop being easily reusable and applicable to multiple situations. The alternative is to make the components easily extendable, by splitting big components in as many independent components as possible, and allowing the user to easily extend and override each component behavior, while matching the same set of requirements.

Our proposal in this work is twofold. We propose a methodology which automates the whole generation process of metaheuristic algorithms from a given set of algorithmic components. To this aim, a classification of components that include all algorithmic elements and their relationships, and a labeling nomenclature that allows the automatic component discovery and analysis is provided, including the automatic grammar generation that models them. Besides, we propose a new parsing and generating strategy for automatically transforming the decisions to the appropriate parameter space as required by the chosen AC tool. The AC tool will be treated as a black-box component, with the ability of being swapped by any other tool if so desired by the user. Moreover, the proposed methodology will be applied to three different combinatorial optimization problems, using a library of already implemented components, starting the automatic configuration from scratch, without providing existing knowledge in the form of initial good configurations. As it will be shown in Section 4, we obtain competitive results in relation to the state of the art.

The rest of the paper is organized as follows: first, we present the full methodology in Section 2. Afterward, in Section 3, the target problems are defined, and a summary of the state of the art is presented. Next, we detail the experimentation methodology in Section 4, and compare the performance of the automatically generated algorithms using the proposed methodology against the state of the art. Finally, the conclusions and future line of research are presented in Section 5.

2. Automatic design by optimization

In this section, the proposed methodology is introduced. As seen in Figure 1, the methodology is formed by two main phases: (1) component discovery and automatic grammar generation, and (2) design by optimization.

Next, we describe the details of both phases of the proposed methodology in a general way, that is, without being bound to a particular algorithm or problem, which is one of the main contributions of this work.

2.1. Components discovery and grammar processing

The first phase of the methodology is implemented by three different steps: the code *scanner*, the *recursive grammar walk*, and the *parameter space transformer*, as seen on the left side of Figure 1. In brief, the methodology has been implemented on different software modules which interact with both the source code that implements the algorithmic components and the selected AC tool.

Before detailing their behavior, some definitions are introduced in the next subsection.

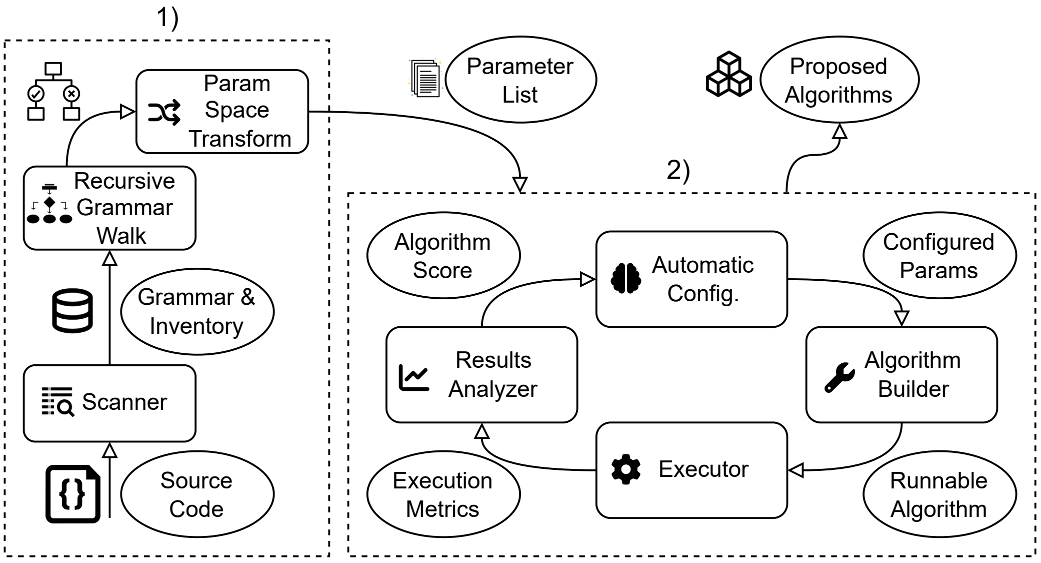


Figure 1: Complete picture of the automatic generation and configuration of algorithms methodology.

2.1.1. Components and relationships

For the sake of generalization, we define *algorithmic component* as any procedure, method or parameter used inside any heuristic or metaheuristic method, and any of their internal components. Examples of algorithm components are constructive methods, stopping conditions, neighborhoods and local search methods. Note that all heuristic and metaheuristic methods are considered as algorithmic components themselves, as they can be used by other metaheuristics.

Any algorithmic component can declare a dependency on any other component type. This dependency will be automatically resolved to any available component which fulfills the required functionality. In this regard, if a given component called *SimpleAlgorithm* requires a component of type *Constructive* in order to work, this dependency can be satisfied by any component that matches the same specification defined by *Constructive* by means of inheritance, composition or any other available mechanism of the programming language used. In Grammar 1, this dependency example is described. The first grammar rule is the list of available algorithms in the current context, represented by the starting symbol S, with each found algorithm as a derivation. The second rule in the example specifies that the algorithm *SimpleAlgorithm* has a dependency on both a *Constructive* and an *Improver* component. *Constructive* can be subsequently replaced with any component that matches its specification, for example, a random shuffle (*RandomShuf-*

fle) of the instance data, or a greedy constructive heuristic (*Greedy*).

$$\begin{aligned}
\langle \text{S} \rangle &::= \langle \text{SimpleAlgorithm} \rangle \mid \ldots && (1)\\
\langle \text{SimpleAlgorithm} \rangle &::= \langle \text{Constructive} \rangle\ \langle \text{Improver} \rangle && (2)\\
\langle \text{Constructive} \rangle &::= \langle \text{RandomShuffle} \rangle \mid \langle \text{Greedy} \rangle && (3)
\end{aligned}$$

Grammar 1: Example grammar with a single algorithm and two constructive implementations.

A set of algorithmic components have been already implemented in a publicly available framework[1] in order to prove our methodology. Besides, any new component coded by a researcher will be automatically discovered by this methodology. Figure 2 shows an example of algorithmic components that could be used to solve the classic Travelling Salesman Problem (Gavish and Graves, 1978). The yellow background components are those provided by our framework, while the blue background ones are custom components developed by a researcher: a custom algorithm, using a memetic metaheuristic (Neri et al., 2011); a constructive method which shuffles the order in which the destinations are visited; a candidate list manager for the GRASP constructive (Feo and Resende, 1989); a swap neighborhood which changes the order in which two destinations are visited, and a Tabu improvement strategy (Glover and Laguna, 1997). Notice the inheritance ("is a") and composition ("part of") relationships, denoted with white triangle and black diamond symbols, respectively.

2.1.2. Code analysis and grammar generation

Under this scenario, the source code is analyzed by the *Scanner* module (see Figure 1), looking for both algorithmic components provided by the framework and those provided by the user. Each component is individually analyzed, listing its dependencies and the roles that it may perform, generating a graph similar to Figure 2. The dependencies are generated by following the principles of OOP (Object-Oriented Programming) languages, where the roles a component may perform are determined by the union of its relationships. For a full technical description of how the implementation of this step works, see Appendix B.

Note that the color differentiation in Figure 2 is only for explanation purposes, as that differentiation does not exist at runtime. This approach allows

[1]https://github.com/mork-optimization/mork

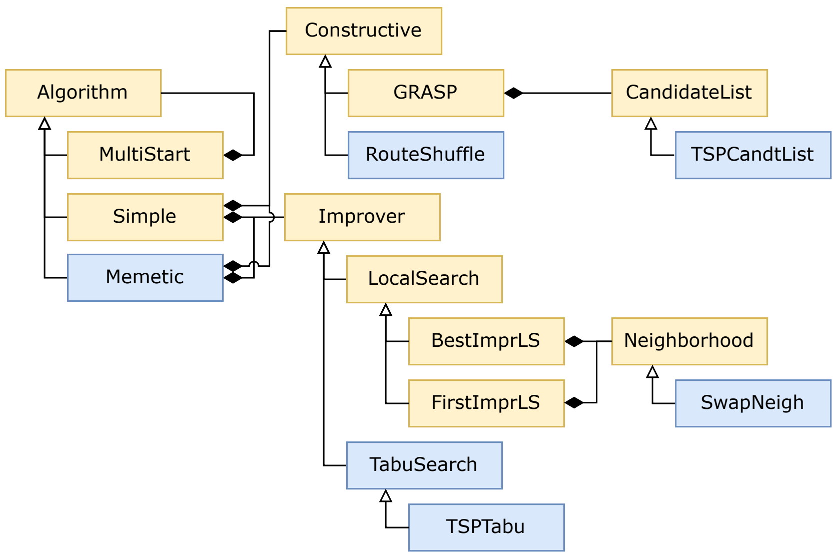


Figure 2: Example of automatically discovered components and the hierarchy they form. Diamond arrows mean that the component at the side of the diamond is composed by or depends on the component at the other side, while the unfilled triangle represents that a given component implements or is of the type pointed by the triangle.

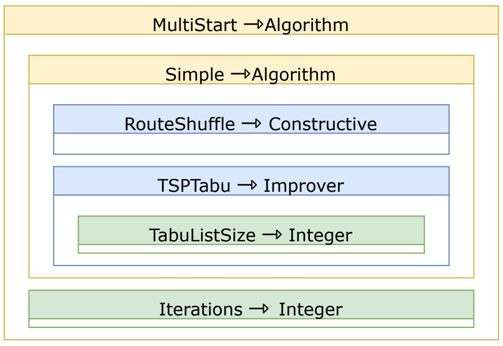
the user to design an arbitrary hierarchy of components and their implementations, that may or may not be related to the existing hierarchies proposed by our framework, making it trivially to extend. The only component that must always be used or inherited is the *Algorithm* component, as it is the starting point for the generated grammar, but it may have dependencies on algorithmic components unknown by the framework, and it will still work, as the dependency graph between components is built at runtime. See Figure A.7 in Appendix A for a full list of the proposed components and types in our framework.

This design strictly adheres to the open-close principle (Martin, 1996b): algorithmic components are open to be extended, but cannot be modified. Or more specifically, users can add their own components, or extend existing ones to add or change their behavior, but they cannot modify their original specification. The importance of following this design is highlighted in (Swan et al., 2019), where it stipulates the requirements for reusing algorithm components in the context of automated assembly of algorithms.

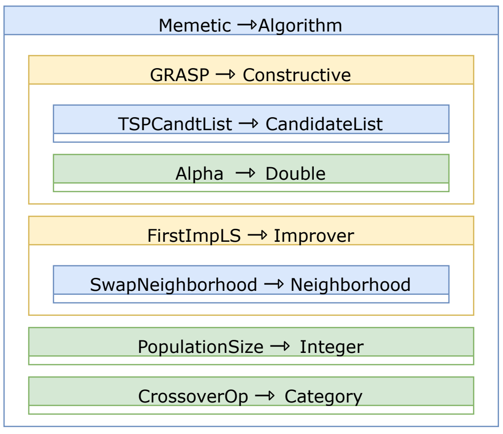
Once all components have been analyzed, and the implicit grammar is generated, the next step consists in exploring all valid component combinations available, or in other words, building the derivation tree. Exploration depth is limited by a maximum depth parameter defined by the user. If a given component combination does not generate a valid proposal, because a component has a dependency that cannot be satisfied, or the maximum depth is reached for a given component combination, the affected branches of the derivation tree are pruned.

The derivation tree defines the space of valid algorithmic component combinations that will be explored by the AC tool. Hence, by means of the derivation tree, any valid algorithm configuration can be generated.

Figure 3 shows two algorithms that could be generated for the example presented in Figure 2. In the first one, Figure 3a, the *MultiStart* algorithm provided by the framework has been chosen, using the *Simple* component as its internal algorithm. The *Simple* algorithm consists in sequentially building a solution using a constructive method, using an improvement method over the generated solution afterward. Parameters are represented as green empty boxes. In Figure 3b, a more complex example is built using the user provided *Memetic* algorithm. Notice that framework provided components and user defined components are intermingled, without any differentiation, which is one of the novelty contributions of our approach. This is demonstrated by the fact that an existing *GRASP* constructive component is chosen, using the user provided list manager.



(a) Example of a generated multi-start GRASP algorithm.



(b) Example of a generated hybrid evolutionary algorithm, using a memetic approach.

Figure 3: Examples of automatically generated algorithms. Each component and parameter is represented as a box, which may recursively contain components. Provided components in the framework are yellow; integer, real, categorical and numerical parameters are green, and blue components are the ones provided by the user

Recursivity is implicitly allowed in our proposal, and may occur in cases when an algorithmic component ends up depending on another component of its very same type or subtype. Figure 2, shows that the *MultiStart* component includes an *Algorithm* component. This is the consequence of the design of *MultiStart*, which executes any component of type *Algorithm* n times. Therefore, it may contain another *MultiStart* component with a different configuration. This behavior can be further tuned using metadata provided in the components, as will be shown in Section 2.2.

Lastly, since the proposed methodology uses an external AC tool to determine the best configuration, the derivation tree is translated into the particular configuration format required by the AC tool. This is implemented by the *Parameter Space Transformer* component, which is dependant on the chosen AC method.

2.2. Automatic design by optimization

The automatic design by optimization proposal is based on the idea that the different algorithmic combinations or configurations correspond to solutions in the space defined by the previously generated grammar. Hence, a process must be initiated where the selected AC tool will explore this solutions space in order to find the best algorithm configuration. In addition, the interaction between the AC tool and the framework modules have to be defined.

2.2.1. Parameter optimization loop

As explained before, the derivation tree generated after the grammar provides the definition of the solution space that feeds the AC. Besides, the AC has to be able to build and run the generated algorithm configuration obtaining a quality value, also named score, to this configuration. This process, represented as a loop in Figure 1, is formed by four distinct independent components:

- **Automatic Config**. Represents the automatic configuration tool, external to the framework. Outputs run configurations, formed by an instance and a set of parameters and their values to test. Receives the score of the given combination of parameters, and decides subsequent parameter combinations to test.

- **Algorithm Builder**. Transforms the parameters as received from the AC method to an intermediate language that is later parsed to build, in execution time, the algorithm requested by the AC. In our implementation, no compilation is needed in this step. Parsing is implemented using ANTLR Parr and Quong (1995). For a complete technical description of this step, see Appendix B.

- **Executor**. Launches the runnable algorithm generated by the algorithm builder using the chosen instance, in a reproducible environment.

Two implementations are provided: the first one executes built algorithms sequentially, while the second one can parallelize a limited set of the runs. All algorithm configurations have a configurable maximum runtime, after which the configuration is terminated if they have not finished earlier.

- **Results analyzer**. Collects the metrics generated during the algorithm execution, specifically how the objective function evolves over time. Typically, the AC tool expects a single number representing the performance of the given parameter combination, so we propose using the AUC (Area Under Curve) metric in order to represent the anytime algorithm performance. For more information about anytime optimization and its relation to the automated configuration of algorithms, see López-Ibáñez and Stützle (2012b).

This process is repeated until the stopping criteria of the AC tool is met, which, except in trivial cases where the full parameter space can be explored, occurs when the tuning budget is exhausted. The best combinations of algorithmic components and their parameters are returned, along with a report summarizing the findings.

2.2.2. Integration with automatic configuration tool

In order to ensure the modularity of our approach, we have created an intermediate language and defined a set of annotations to describe parameters, that greatly simplifies integration between any AC tool and the framework, by allowing us to define arbitrary algorithmic component combinations and their respective parameters as simple strings. Moreover, this representation improves the explainability of the configurations as they are being tuned, and greatly simplifies debugging when compared against the equivalent AC output.

Six different parameter types are proposed to describe all possible components in the methodology and their dependencies. These parameter are:

- *Algorithmic Component parameter*: represents any algorithmic component type required as a dependency. By default, any component that matches the given type could be used. For example, a parameter of type *Improver*, could be filled either by any *LocalSearch* component, or any custom *Improver* implemented by the user. An algorithmic component may optionally provide metadata to explicitly declare which components are allowed, or block some specific components, reducing the number of possibilities available by default. For example, it may be desired to avoid using a *Multistart* inside another *Multistart* component.

- *Context parameter*: represents any parameter type whose value is either fixed or calculated at runtime by a user provided function. Examples can be the algorithm name, or the direction of the objective function (maximize or minimize). Independently of their type, the automatic configuration engine will not consider them as parameters to be passed to the AC tool, and will be automatically filled by the algorithm builder component when generating the algorithm using the parameters provided by the AC tool.

- *Integer parameter*: represents an integer value, allowing the user to define a range of valid values that the component accepts.

- *Real parameter*: represents a real value, allowing the user to define a range of valid values that are valid for the given component.

- *Categorical and Ordinal parameters*: represents a decision to be taken between a predefined set of values. An example of a categorical parameter could be the crossover operator type used in a population-based algorithmic component. If there is an implied order between the possible values, the parameter type is called ordinal.

Using the derivation tree built at the end of the first phase, we can always generate a parametric description with the proposed intermediate language (López-Ibáñez et al., 2016). The parameter description will use all found parameters found in each component, in addition to categorical parameters to represent choices among components, using conditions to control which other components (and their parameters) are activated when certain components are selected earlier in the derivation tree.

A parametric description of the design space enables the use of off-the-shelf automatic algorithm configuration methods, such as `irace` (López-Ibáñez et al., 2016). We have chosen `irace` as the AC software, due to the fact that it is open source, well documented, widely used and matches all our requirements. Moreover, previous benchmarks and research has shown it is very capable (Rasku et al., 2019).

Listing 1 shows an example of a run request to our framework from the `irace` AC tool. The parameters in the request are transformed to our intermediate language as shown in Listing 2.

Listing 1: Example command line call made by `irace` to execute the algorithm presented in Figure 3a.

```
--Algorithm=MultiStart --Algorithm_MultiStart.algorithm=Simple
   ↪ --Algorithm_MultiStart.iterations=10 --
   ↪ Algorithm_MultiStart.algorithm_Simple.constructive=
   ↪ RouteShuffle --Algorithm_MultiStart.algorithm_Simple.
   ↪ improver=TSPTabu --Algorithm_MultiStart.algorithm_Simple
   ↪ .improver_TSPTabu.tabuListSize=250
```

Listing 2: Intermediate language representation of the algorithm presented in Figure 3a

```
MultiStart{
  iterations=10,
  algorithm=Simple{
    constructive=RouteShuffle{},
    improver=TSPTabu{
      tabuListSize=250
    }
  }
}
```

However, due to the fact that the AC component is loosely coupled to the framework, other tools may be easily integrated, such as Optuna (Akiba et al., 2019), SMAC (Hutter et al., 2011) or ParamILS (Hutter et al., 2009). Additionally to the parameter list using the format required by the chosen tool, a configuration file is provided specifying, among others, the tuning budget, which is calculated dynamically according to the total number of parameters to tune, scaling accordingly automatically.

3. Target problems

The proposed methodology for the automated design of metaheuristics will be tested on three optimization problems from different problem families. In this section, each problem is introduced along with a summary of its state of the art.

3.1. Space-Free Multi-Row Facility Layout Problem (SF-MRFLP)

The Space-Free Multi-Row Facility Layout Problem (SF-MRFLP) belongs to the family of Facility Layout Problems (FLP). This family encompasses a set of optimization problems devoted to placing facilities using a given layout as the main constraint, trying to optimize magnitudes that are usually related to handling material among different facilities or operation costs (Anjos and Vieira, 2017). Examples of layouts are circular or loop layouts, row based layouts and open-field layouts. The SF-MRFLP generalizes the Corridor Allocation Problem (CAP) (Amaral, 2012), in which only two rows may be used.

The SF-MRFLP consists of placing a predefined number of facilities in a layout formed by two or more rows, taking into account that no free space is allowed between adjacent facilities in the same row, and that all rows must be aligned to the left-hand side. Each facility i has a particular length l_i and a list of weights w_{ij} specifying the material flow to other facilities j. The position of each facility i in its assigned row determines its loading point x_i, which is situated at the center of the facility. The loading point is calculated as the sum of the lengths of all facilities before itself, plus half its length. If the facility is the first in the row, the loading point is simply $x_i = \frac{l_i}{2}$. The goal is to minimize the *material handling cost* (MHC), which is calculated as the sum of the horizontal distances, assuming the distances between rows and the height of the facilities are negligible, between the loading points of each facility pair, multiplied by their respective material flow:

$$\text{Minimize} \quad \sum_{i=1}^{n} \sum_{j=i+1}^{n} w_{ij} \cdot |x_i - x_j| \tag{4}$$

Several authors have addressed this problem. To the best of our knowledge, the state of the art metaheuristic is a variable neighborhood search (VNS) described by Herrán et al. (2021). The method uses a GRASP approach for the constructive phase, using two different strategies for picking elements from the candidate list, *RandomGreedy* and *GreedyRandom*. During the improvement step of the VNS algorithm, three different neighborhoods are used: removing and adding a facility from one position to another, swapping two facilities, and an extended neighborhood formed by both. Furthermore, the proposed neighborhoods can be explored using three different strategies: first improvement, best improvement and hybrid (pick best move in a smaller section of the neighborhood).

3.2. Balanced Minimum Sum-of-squares Clustering problem (BMSSC)

The second problem used to validate the proposal is the Balanced Minimum Sum-of-squares Clustering (BMSSC) problem (Xavier and Xavier, 2011). This problem is devoted to assigning a number of elements into sets (clusters) of equal size. The number of clusters k is given *a priori*. If the number of elements n is not divisible by k, there will be $n \bmod k$ clusters of size $\lceil n/k \rceil$, and $k-(n \bmod k)$ clusters of size $\lfloor n/k \rfloor$. The objective function, represented in Equation 5, consists in minimizing the sum-of-squares distance of an assignment of points $P = \{p_1, p_2, \ldots, p_n\}$, located in a Euclidean space, to the k clusters, taking into account the distance of each point p_j to the centroid c_i of the cluster it is assigned to. The centroid c_i is defined as the point with the minimum distance to all points assigned to the same cluster i. a_{ij} is a binary variable that takes the value of 1, if point j is assigned to cluster i, and 0 in all other cases.

$$\min \sum_{i=1}^{k} \sum_{j=1}^{n} ||p_j - c_i||^2 \cdot a_{ij} \tag{5}$$

The state of the art is Martín-Santamaría et al. (2022), where the authors propose combining a GRASP approach (Feo et al., 1994) with the Strategic Oscillation (SO) method (Glover and Hao, 2011). Two neighborhoods are proposed: the first one swaps elements between clusters, which maintains solution feasibility; while the second one removes and element from its currently assigned cluster and reassigns it to a different one, which due to the problem restrictions is likely to break the size constraint. The authors propose using the first neighborhood during the local search phase of the GRASP metaheuristic, while using the second one during the strategic oscillation step, while relaxing the cluster size constraint. Afterward, the solutions may need to be repaired, which is achieved by using the same neighborhood to reassign elements from overloaded clusters to undersized clusters.

3.3. Vehicle Routing Problem with Occasional Drivers (VRPOD)

The third and final problem belongs to the family of the Vehicle Routing Problems (VRP). The main objective of the VRP is to create delivery routes, in order to fully satisfy consumer demand, while minimizing operating costs. The VRPOD, firstly introduced in Archetti et al. (2016), is a variant of the VRP that considers the possibility of having occasional drivers (ODs) to attend some customers, therefore reducing the length and number of routes, and the associated operating costs. An occasional driver is not a professional driver, they may be a customer that goes to a physical shop location, and may agree to deliver a package from an online order for a small compensation if the package destination is in route to their destination. The objective function takes into account both the cost of the vehicle fleet and the sum of compensations to the occasional drivers. Different compensation schemes may be used, see Archetti et al. (2016) for more details and the full problem formulation.

The current state of the art for the VRPOD is Martín-Santamaría et al. (2021), where the results obtained by Archetti et al. (2016) are improved. The authors propose using a cooperative parallel Iterative Local Search Scheme, proposing five different neighborhoods for the local search step: three of which are commonly used in VRP problems, specifically move one package from one route to another one in any position, swapping two deliveries, and reversing a route fragment, also known as 2-Opt. Moreover, two specific neighborhoods for the VRPOD related to the usage of ODs are used: assigning a delivery to an OD, and therefore removing it from its associated route, or the reverse operation, removing an OD and assigning its delivery to any route in any position. For the perturbation step, three different methods are presented: removing a random route, using a probability distribution which depends on the total cost of the route; randomly deassigning a percentage of all deliveries, to be later repaired; and randomly apply movement from any neighborhood.

4. Experimental analysis

In this section, the experimentation environment is firstly detailed. Then, the modifications performed to the source code and its rationale are presented. In order to validate that our modifications do not affect the original proposals' performance, a comparison will be made against the modified code, which we will call *reimplementation* in this section. After the reimplementation has been validated, the automatic design experiment is detailed, and the best configurations found will be compared against the reimplementation.

4.1. Reimplementation of previous methods

Due to the fact that different problems are implemented in different languages, using different hardware to execute the experimentation, and with different parallelization strategies, the first step we have taken is to create a common environment. Specifically, all algorithms have been reimplemented in Java 17. The JVM has been limited to 4GB of RAM, and all experiments are run in a single thread configuration. All experiments have been executed in VMs, running on a cluster formed by multiple hosts with 2x AMD EPYC 7282 CPUs and 96GB of RAM.

The state-of-the-art problems have been reimplemented in Java 17, splitting the code into independent components following the framework design principles, as presented in Section 1. The only components from the state-of-the-art proposals that are not ported due to technical complexities are those responsible for parallelizing the corresponding previous approach. If there were hardcoded parameters in the original code, the code has been modified to accept different values in reasonable ranges. In the first experiment, when comparing the reimplementation performance against the previous authors code, the original parameter values are used.

In order to validate our reimplementation of the previous methods against the source code provided by the previous authors, we have executed 30 times both the actual previous approach from the state of the art as is and our reimplementation, using as stopping condition the limit proposed in the respective state-of-the-art paper, be it a time limit or a maximum number of algorithm iterations, being specific for each one of the three problems. While the resulting execution times are different, as would be expected due to using a different experimental environment, the gap between the objective function values is very small. In particular, the percentage average difference in objective function values between the state of the art and the reimplementation is 0.02% for the SF-MRFLP, 0.12% for the BMSSC and 3.93% for the VRPOD. Notice that the slightly higher percentage average in the last case can be explained by the absence of the parallel cooperative scheme in the reimplementation, as detailed before.

4.2. Automated Metaheuristic Design

The methodology and configuration designed for automatically designing metaheuristics is exactly the same for the three target problems. We have chosen `irace` as the AC tool, scaling the computational budget dynamically according to the number of parameters. Specifically, `irace` will have a budget of 10 000 evaluations per 50 parameters, with a minimum of 10 000, where each evaluation corresponds to running once one candidate design on one training instance. For example, in the case of SF-MRFLP, the design space has a total of 203 parameters, so `irace` will execute a maximum of 40 600 evaluations. In the case of the BMSSC and the VRPOD, the number of parameters is 261 and 149, respectively.

The objective function used in `irace` to optimize algorithm configurations is the AUC (Area Under Curve) (López-Ibáñez and Stützle, 2014) for the objective function of the given problem, measured over time, discarding the area outside the interval $[10, 60]$ in seconds. Any algorithm configuration that does not report a score value before the first 10 seconds is considered invalid and removed. `irace` is configured to evaluate the configurations' performance using the Friedman test, and to start from randomly sampled algorithm designs without any prior knowledge about good algorithm designs or default parameter values, which demonstrates one of the main contributions of the paper. For each studied problem, the best algorithm design found by `irace` will be compared with the validated reimplementation.

4.3. Comparison with reimplementation

Once an algorithm configuration has been obtained for each problem, the comparison with the reimplementation of the state-of-the-art method will use the same methodology as when validating the reimplementation. In order to perform a fair comparison, for each instance of the target problem, the automatically generated configuration, called *Auto*, will be executed 30 times using as time limit the average execution time spent by the reimplementation for each instance in the previous experiment.

For each problem, the following four metrics are reported, comparing the reimplementation and *Auto*: number of times that the configuration reaches the best known value for all instances (#Times reaches *bkv*); number of instances for which the algorithm obtains the best known value at least once (#Instances finds *bkv*); averaged percentage deviation of the best value found by the configuration to the best known value (%Dev Min); and lastly, averaged percentage deviation of all iterations to the best known value (%Dev Avg).

Moreover, with the objective of statistically assessing the performance of the *Auto* approach with respect to the reimplementation, we will use the Bayesian performance analysis over all the instances simultaneously described in Calvo et al. (2019). In this approach, the algorithms are ranked per instance and the expected winning probability for each algorithm is computed. Moreover, it is also useful for assessing the estimation uncertainty using credible intervals. In Bayesian statistics, credible intervals estimate the range of values the unknown parameter may have with a given probability.

In the next subsections, we summarize the results obtained for each problem. For a more detailed analysis, including descriptions of the generated configurations, full tables, generated figures, etc., see Appendix C, where references to Zenodo and code repositories for each problem are provided.

4.4. Results for the SF-MRFLP

	$Auto_{SF\text{-}MRFLP}$	Reimplementation
#Times reaches *bkv*	485 (19.96%)	895 (36.83%)
#Instances finds *bkv*	78 (96.30%)	42 (51.85%)
%Dev Min	0.00	0.01
%Dev Avg	0.45	0.03

Table 1: Comparison between the automatically generated algorithm and the reimplementation of the state-of-the-art for the SF-MRFLP. *bkv* are the initials of best known value.

Table 1 shows the comparison between the best algorithm found by the automatic configuration procedure, called $Auto_{SF\text{-}MRFLP}$ algorithm with the reimplementation of the state-of-the-art method. We can observe that the reimplementation obtains the best known value around 37%, while $Auto_{SF\text{-}MRFLP}$ obtains about 20%. However, the $Auto_{SF\text{-}MRFLP}$ algorithm reaches the best known value for 78 out of 81 instances, while its counterpart reaches it only for 42 instances. This difference in results can be explained by the fact that, in most cases, the reimplementation either reaches the best value for an instance in all iterations, or none, as can be seen in the detailed results provided as complementary material. The deviations show us the same behavior. While the $Auto_{SF\text{-}MRFLP}$ reaches at least a better value once, the original proposal obtains slightly better results on average.

Regarding the statistical analysis, Figure 4 shows, with a green dot, the average probability of winning obtained by the Bayesian approach previously described. We can observe in the plot that $Auto_{SF\text{-}MRFLP}$ has a greater average chance of winning than the reimplementation. However, since the confidence intervals overlap, the performance of both algorithms could be equivalent in some instances. In other words, the $Auto_{SF\text{-}MRFLP}$ algorithm reaches the performance of the manually-designed state-of-the-art algorithm despite being generated automatically from a large space of algorithmic components without any expert guidance.

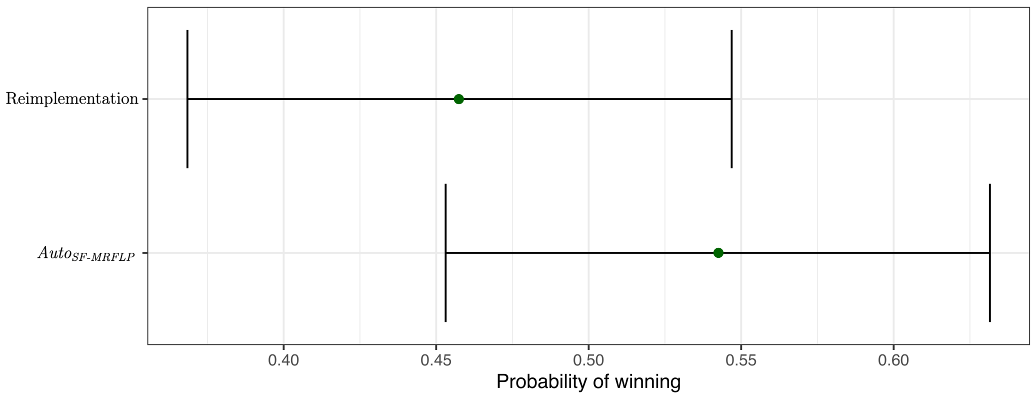


Figure 4: Expected winning probability for both the SF-MRFLP state-of-the-art and the best automatically generated algorithm found.

4.5. Results for the BMSSC

Table 2 shows the comparison between the best automatically generated configuration found, called $Auto_{BMSSC}$, and the reimplementation proposals. We can see that both the number of times the best value is reached and the unique instances for which the $Auto_{BMSSC}$ has the best known value is slightly higher. Moreover, the $Auto_{BMSSC}$ proposal obtains a lower deviation minimum deviation (0.03%) to the best known values, and an equal average (0.46%).

With regard to the statistical analysis in Figure 5, while the $Auto_{BMSSC}$ has a slightly greater chance of winning (52% vs 48%), the confidence intervals are overlapped, so we can consider the performance of both algorithms to be similar.

	$Auto_{BMSSC}$	Reimplementation
#Times reaches *bkv*	221 (29.47%)	206 (27.47%)
#Instances finds *bkv*	18 (72.00%)	17 (68.00%)
%Dev Min	0.03	0.20
%Dev Avg	0.46	0.46

Table 2: Comparison between the automatically generated algorithm and the reimplementation of the state-of-the-art for the BMSSC. *bkv* are the initials of best known value.

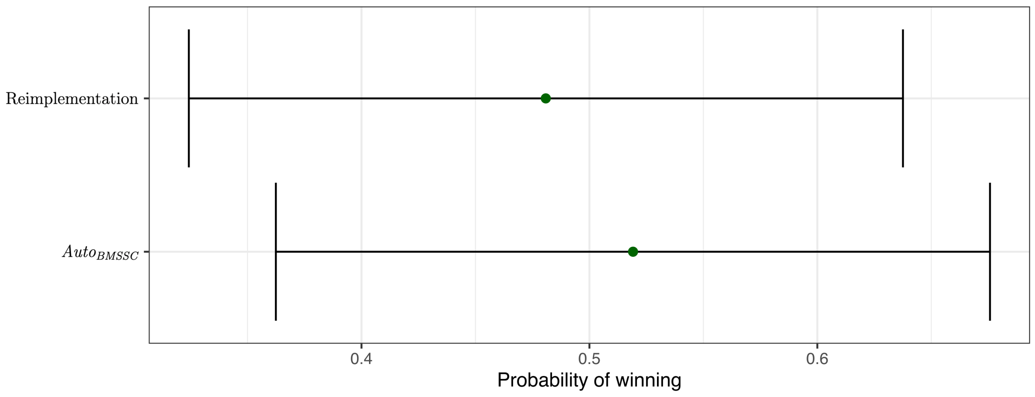


Figure 5: Expected winning probability for both the BMSSC state-of-the-art and the best automatically generated algorithm found

4.6. Results for the VRPOD

In Table 3 we can observe the comparison between the automatically generated configuration, called $Auto_{VRPOD}$ and the reimplementation proposals. In this case, the $Auto_{VRPOD}$ reaches the best known value around 30% of times, having the best value for 89% of instances, while the reimplementation obtains 19% and 31% respectively. However, both deviations are worse for the $Auto_{VRPOD}$. This can be explained by the fact that for about 1%

VRPOD	$Auto_{VRPOD}$	Reimplementation
#Times reaches *bkv*	3816 (30.29%)	2373 (18.83%)
#Instances finds *bkv*	372 (88.57%)	132 (31.43%)
%Dev Min	1.55	0.68
%Dev Avg	5.06	1.91

Table 3: Comparison between the automatically generated algorithm and the reimplementation of the state-of-the-art for the VRPOD. *bkv* represents best known value.

percentage of the instances, the minimum deviation is greater than 10%, and around 6% for the average deviation.

Regarding the statistical analysis in Figure 6, we can observe that the $Auto_{VRPOD}$ has a much stronger probability of winning, with non overlapping intervals. We can assert with confidence that the performance of the $Auto_{VRPOD}$ proposal is better than the Reimplementation.

The big difference of results obtained by both methods, comparing the results with the other two problems, may be explained by the types of instances. Instances in the VRPOD can be divided according to the compensation scheme used to pay the occasional drivers. Analyzing the results, we have observed that the $Auto_{VRPOD}$ proposal obtains good results for both compensation schemes types available in the state of the art, while the reimplementation performs well for the first compensation scheme but falls behind in the second type. Due to this, we may conclude that the $Auto_{VRPOD}$ generalizes better the instances properties, and therefore it is much more likely to win under new instance sets.

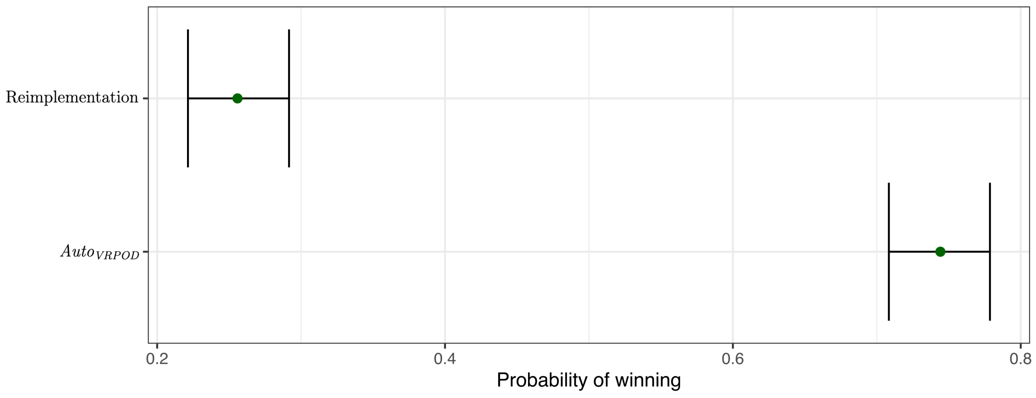


Figure 6: Expected winning probability for both the VRPOD state-of-the-art and the best automatically generated algorithm found

To sum up, the automatic configuration methodology is able to find algorithms with equal or greater performance than traditional manual tuning, in an automatic and reproducible way.

5. Conclusions and Future Work

In this work, we have proposed a methodology to automatically generate metaheuristic approaches for optimization problems, taking special care of the research reproducibility.

We have demonstrated the advantages of the proposed methodology, improving in key areas over existing works, notably proposing a replacement for the manual preliminary experimentation approach commonly used via the use of automatic algorithm component discovery and their configuration. Moreover, we have demonstrated how automatic configuration proposals can improve or at least tie with existing results, using three problems from completely different problem families, a facility layout problem, a vehicle routing problem and a clustering problem.

The algorithms generated for each problem start with no previous knowledge of the performance of each component. An idea worth exploring in future works may be to analyze the effect of providing good known existing configurations, such as those existing in the state of the art, and measure how they affect the performance and convergence of the optimization engine.

Another aspect that needs further exploration is analyzing the algorithm equivalencies and investigating the viability of automatically simplifying configurations generated by the AC tool, and immediately return if it is found that it is equivalent to an already evaluated configuration. An example may be a *MultiStart* algorithm with a number of executions $n = 1$, where removing the *MultiStart* component and calling the internal algorithm directly should obtain the same results.

Appendix A. Full component hierarchy

In this Annex, an overview of the full component hierarchy is provided.

Figure A.7 shows the hierarchy formed by the common components used in the experiments, based on the usual classification of algorithms: constructive methods, improving methods, perturbation methods and neighborhoods. Empty triangle arrows represent the type hierarchy, or in other words, where a given component may be used. For example, both *MultiStart* and *Simple* are of type *Algorithm*, and may be used anywhere where an algorithm is requested. Furthermore, diamond arrows mean that the component at the side of the diamond is composed by or depends on the component at the other side, that is, a *MultiStart* element is composed by an *Algorithm* component and *Simple*, *IteratedGreedy*, *VNS* and *SimAnnealing* algorithms need a *Constructive* component, for instance. As it can be seen in the blue elements, user components may extend or implement any functionality to match its needs, and the components are automatically detected and added to the hierarchy.

Components are ordered in three columns to facilitate the explanation. The first column, represents the *Algorithm* component type, which is the only mandatory one that must be always used, either by extending it and adding a custom implementation, or by using any of the already implemented components. In the second column, we see the most common components type used in the literature, namely *Constructive*, *Improver* and *Shake* or perturbation components. Finally, in the third column, we have any specific components required depended on by any other component, or problem specific components. We are continuously working on expanding the hierarchy in order to include more metaheuristic approaches. See the GitHub repository for the latest code `https://github.com/mork-optimization/mork` and the docs at `https://docs.mork-optimization.com` for a more up-to-date version.

Several examples of recursivity can be seen in Figure A.7, for instance the *MultiStart* algorithm requires an algorithm over which it will iterate. Another example is the *VND* Duarte et al. (2018), which consists on trying multiple improve methods on a deterministic way, and therefore needs multiple *Improver* methods. One of this improvement methods by default may be another *VND* with a different configuration. If recursion is not desirable for a given component, it can be specified as metadata inside the component, or checked when constructing the component and removing the candidate configuration due to a broken constraint.

Appendix B. Technical implementation details: annotations and intermediate language

In order to easily allow users to specify the components or parameters metadata, we propose to the use of a kind of "marks" inside the user code that do not affect how it works. Most programming languages allow a way to declare this kind of marks that can provide arbitrary metadata and automate their parsing either during compilation or at runtime. Most importantly, marks do not affect the code behavior. Implementation examples of this pattern in commonly used programming languages are as follows:

- **Java** - Known as annotations. See `https://docs.oracle.com/javase/tutorial/java/annotations/` for more information.

- **Python** - Known as decorators. `https://peps.python.org/pep-0318/`

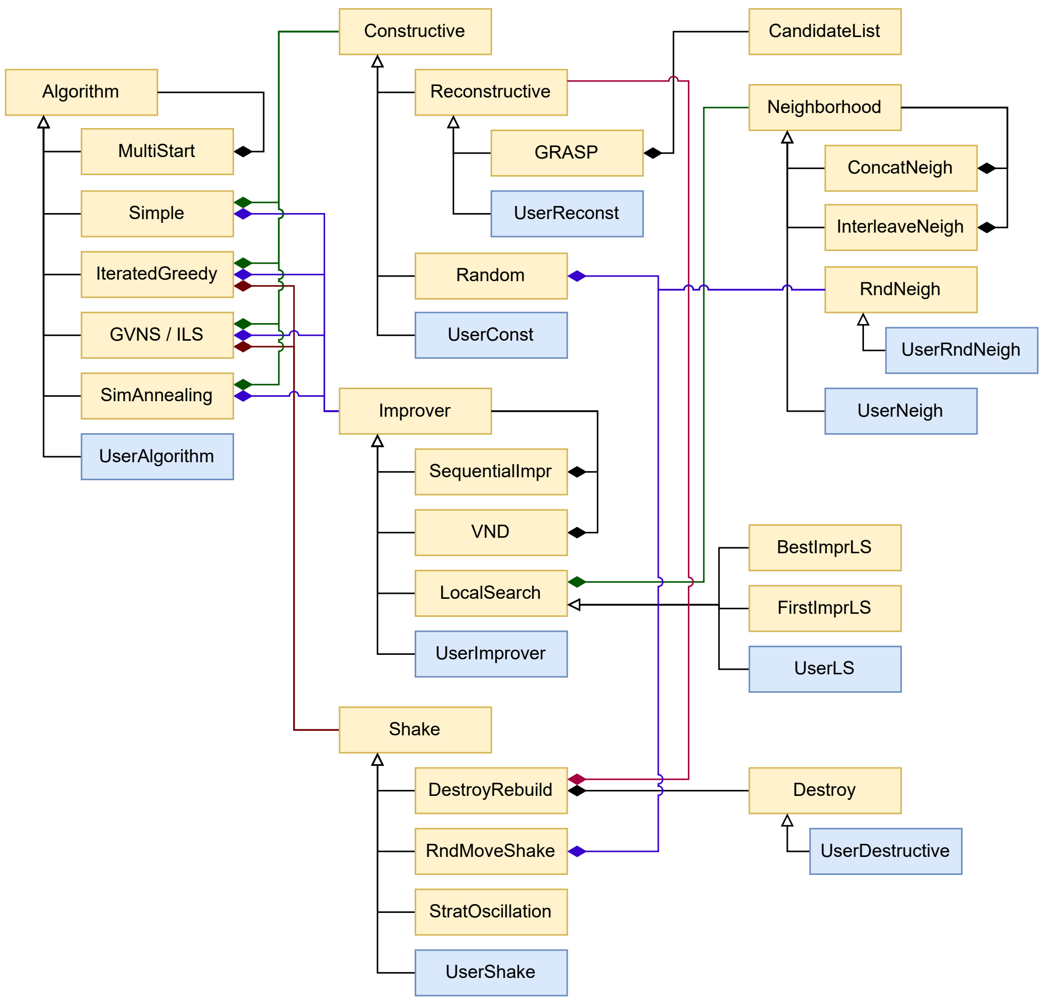


Figure A.7: Algorithm component hierarchy provided by the framework. Diamond arrows mean that the component at the side of the diamond is composed by or depends on the component at the other side, while the unfilled triangle represents that a given component implements or is of the type pointed by the triangle. Colors are used to more easily differentiate relationships.

- **C#** - Known as attributes. See https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/attributes/ for more information.

Although the framework has been implemented using the Java programming language, the same concepts can be applied to most programming languages. An example of an annotated component in Java can be seen in Figure B.8. As Java supports multiple constructor methods in the same class, the first annotation, `@AutoconfigConstructor`, is used to differentiate which constructor should be preferred when automatically building the component. The first parameter, `name`, is annotated as a `@ProvidedParam`, which prevents the parameter from being passed to the parameter optimizer, and a strategy will be found at runtime to fill its value. In this case, a random name will be generated for each configuration of this algorithm. For the second and third parameters, the annotation `@IntegerParam` is used to restrict the parameter's maximum and minimum values. These restrictions will then be translated appropriately to the format used by the parameter optimizer. The fourth and fifth parameters are component dependencies without restrictions, and lastly the sixth parameter is a component where the `@ComponentParam` annotation is used to prevent the listed implementations or its children from being used. By default, if no annotation is used, any subclass of `Improver`, or more abstractly, any other component that matches the same specification, could be used.

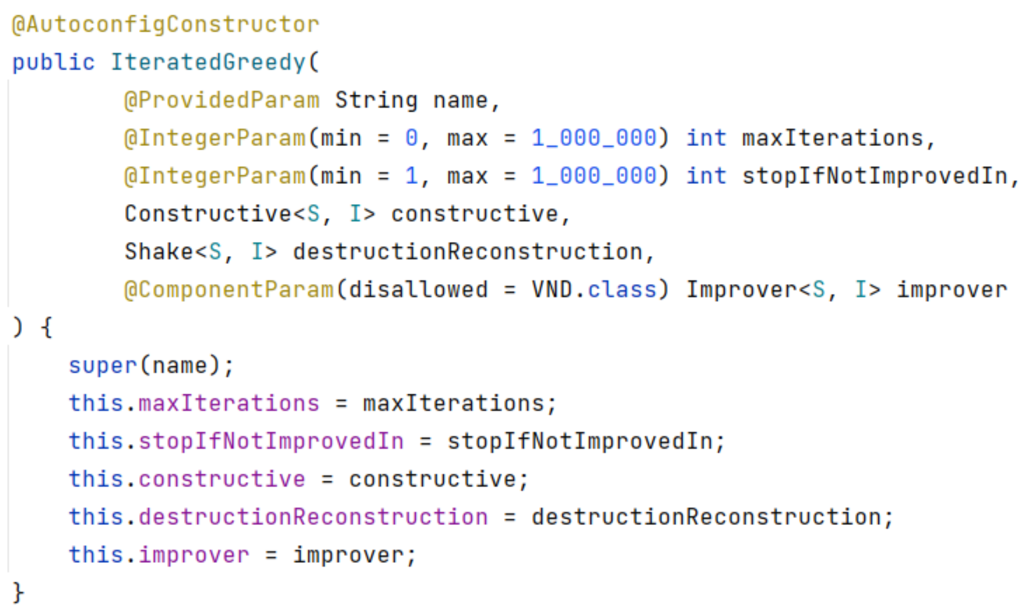
As introduced previously, our implementation uses an intermediate language to represent algorithm configurations, and has a translator to transform them into the format required by the AAC tool, and transform back combinations of parameters to the intermediate language.

This intermediate language allows the usage of any AAC tool, and eases debugging. The language is inspired by both JSON and default Java object to string formatters. In Listing 3, the main rules of the grammar that defines the intermediate language are presented.

Examples of algorithms represented using the intermediate grammar can be seen in the next section, representing the best configurations found for each target problem.

Appendix C. Generated configurations and artifacts

In this Appendix, we will detail the configurations found by the Autoconfig procedure. Configurations are represented using the intermediate language described in the previous section.



```java
@AutoconfigConstructor
public IteratedGreedy(
        @ProvidedParam String name,
        @IntegerParam(min = 0, max = 1_000_000) int maxIterations,
        @IntegerParam(min = 1, max = 1_000_000) int stopIfNotImprovedIn,
        Constructive<S, I> constructive,
        Shake<S, I> destructionReconstruction,
        @ComponentParam(disallowed = VND.class) Improver<S, I> improver
) {
    super(name);
    this.maxIterations = maxIterations;
    this.stopIfNotImprovedIn = stopIfNotImprovedIn;
    this.constructive = constructive;
    this.destructionReconstruction = destructionReconstruction;
    this.improver = improver;
}
```

Figure B.8: Example of the usage of annotations in Java.

Listing 3: Grammar rules of the Intermediate language. The grammar is used to generate a lexer, parser and visitor automatically using the ANTLR library. Semicolons are used to delimit grammar rules, colons separate a grammar non-terminal from its derivations. The vertical bar represents alternative derivations.

```
component: ID '{' properties? '}';
properties: property (',' property)*;
property: ID EQ propertyValue;
propertyValue: literal | component;
literal: NullLiteral | BooleanLiteral | FloatingPointLiteral |
   ↪ IntegerLiteral | StringLiteral | CharacterLiteral |
   ↪ arrayLiteral;
```

Listing 4: Intermediate language representation of the best algorithm found for the SF-DRFLP problem

```
IteratedGreedy{
        maxIterations=535158,
        stopIfNotImprovedIn=664797,
        destructionReconstruction=CAPShake{
                type="1"
        },
        improver=SequentialImprover{
                improverA=CAPLS{
                        type="ins_bi"
                },
                improverB=SequentialImprover{
                        improverA=NullImprover{},
                        improverB=CAPLS{
                                type="exc_bi"
                        }
                }
        },
        constructive=CAPConstructive{
                alpha=0.93,
                type="greedyB2"
        }
}
```

Listing 5: Intermediate language representation of the best algorithm found for the BMSSC problem

```
VNS{
        shake=StrategicOscillation{
                increment=0.45
        },
        maxK=20,
        improver=VND{
                improver1=ShakeImprover{
                        shake=StrategicOscillation{
                                increment=0.33
                        },
                        improver=NullImprover{}
                },
                improver2=FirstImpLS{},
                improver3=BestImpLS{},
        },
        constructive=RandomGreedyGRASPConstructive{
                alpha=0.75{},
                candidateListManager=BMSSCListManager{}
        }
}
```

Listing 6: Intermediate language representation of the best algorithm found for the VR-POD problem

```
IteratedGreedy{
        maxIterations=743168,
        stopIfNotImprovedIn=977131,
        destructionReconstruction=DestroyRebuild{
                constructive=VRPODGRASPConstructive{
                        alpha=0.03
                },
                destructive=RandomDeassign{}
        },
        improver=LocalSearchBestImprovement{
                neighborhood=VRPODExtendedNeighborhood{}
        },
        constructive=VRPODGRASPConstructive{
                alpha=0.06
        }
}
```

Moreover, the links to the source code repositories and all artifacts published in Zenodo are available in Table C.4.

Problem	Live Code	Archived Artifacts DOI
BMSSC	rmartinsanta/ac-BMSSC	10.5281/zenodo.7774638
VRPOD	rmartinsanta/ac-VRPOD	10.5281/zenodo.7774831
SFDRFLP	rmartinsanta/ac-SFDRFLP	10.5281/zenodo.7774833

Table C.4: References to source code repositories and archived artifacts in Zenodo.

References

Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M., 2019. Optuna: A next-generation hyperparameter optimization framework, in: Teredesai, et al. (Eds.), 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM Press, New York, NY, pp. 2623–2631. doi:10.1145/3292500.3330701.

Amaral, A.R.S., 2012. The corridor allocation problem. Computers & Operations Research 39, 3325–3330. doi:10.1016/j.cor.2012.04.016.

Anjos, M.F., Vieira, M.V., 2017. Mathematical optimization approaches for facility layout problems: The state-of-the-art and future research directions. European Journal of Operational Research 261, 1–16.

Archetti, C., Savelsbergh, M., Speranza, M.G., 2016. The vehicle routing problem with occasional drivers. European Journal of Operational Research 254, 472–480. doi:10.1016/j.ejor.2016.03.049.

Cahon, S., Melab, N., Talbi, E.G., 2004. ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics. Journal of Heuristics 10, 357–380. doi:10.1023/B:HEUR.0000026900.92269.ec.

Calvo, B., Shir, O.M., Ceberio, J., Doerr, C., Wang, H., Bäck, T., Lozano, J.A., 2019. Bayesian performance analysis for black-box optimization benchmarking, in: López-Ibáñez, M., Auger, A., Stützle, T. (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference, GECCO Companion 2019. ACM Press, New York, NY, pp. 1789–1797. doi:10.1145/3319619.

Dréo, J., Liefooghe, A., Verel, S., Schoenauer, M., Merelo, J.J., Quemy, A., Bouvier, B., Gmys, J., 2021. Paradiseo: from a modular framework for evolutionary computation to the automated design of metaheuristics: 22 years of Paradiseo, in: Chicano, F., Krawiec, K. (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference, GECCO Companion 2021. ACM Press, New York, NY, pp. 1522–1530. doi:10.1145/3449726.3463276.

Duarte, A., Sánchez-Oro, J., Mladenović, N., Todosijević, R., 2018. Variable neighborhood descent, in: Martí, R., Pardalos, P.M., Resende, M.G.C. (Eds.), Handbook of Heuristics. Springer International Publishing, pp. 341–367. doi:10.1007/978-3-319-07124-4_9.

Feo, T.A., Resende, M.G.C., 1989. A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters 8, 67–71.

Feo, T.A., Resende, M.G.C., Smith, S.H., 1994. A greedy randomized adaptive search procedure for maximum independent set. Operations Research 42, 860–878.

Gavish, B., Graves, S.C., 1978. The travelling salesman problem and related problems. Operations Research Center Working Paper .

Glover, F., Hao, J.K., 2011. The case for strategic oscillation. Annals of Operations Research 183, 163–173.

Glover, F., Laguna, M., 1997. Tabu Search. Kluwer Academic Publishers, Boston, MA, USA.

Grefenstette, J.J., 1986. Optimization of control parameters for genetic algorithms. IEEE Transactions on Systems, Man, and Cybernetics 16, 122–128. doi:10.1109/TSMC.1986.289288.

Herrán, A., Colmenar, J.M., Duarte, A., 2021. An efficient variable neighborhood search for the space-free multi-row facility layout problem. European Journal of Operational Research doi:10.1016/j.ejor.2021.03.027.

Hoos, H.H., Stützle, T., 2004. Stochastic Local Search: Foundations and Applications. Elsevier, Amsterdam, The Netherlands.

Hutter, F., Hoos, H.H., Leyton-Brown, K., 2011. Sequential model-based optimization for general algorithm configuration, in: Coello Coello, C.A. (Ed.), Learning and Intelligent Optimization, 5th International Conference, LION 5. Springer, Heidelberg. volume 6683 of *Lecture Notes in Computer Science*, pp. 507–523. doi:10.1007/978-3-642-25566-3_40.

Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T., 2009. ParamILS: an automatic algorithm configuration framework. Journal of Artificial Intelligence Research 36, 267–306. doi:10.1613/jair.2861.

KhudaBukhsh, A.R., Xu, L., Hoos, H.H., Leyton-Brown, K., 2009. SATenstein: Automatically building local search SAT solvers from components, in: Boutilier, C. (Ed.), Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09), AAAI Press, Menlo Park, CA. pp. 517–524.

KhudaBukhsh, A.R., Xu, L., Hoos, H.H., Leyton-Brown, K., 2016. SATenstein: Automatically building local search SAT Solvers from Components. Artificial Intelligence 232, 20–42. doi:10.1016/j.artint.2015.11.002.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., Birattari, M., 2016. The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives 3, 43–58. doi:10.1016/j.orp.2016.09.002.

López-Ibáñez, M., Stützle, T., 2012a. The automatic design of multi-objective ant colony optimization algorithms. IEEE Transactions on Evolutionary Computation 16, 861–875. doi:10.1109/TEVC.2011.2182651.

López-Ibáñez, M., Stützle, T., 2012b. Automatically Improving the Anytime Behaviour of Optimisation Algorithms. Technical Report TR/IRIDIA/2012-012. IRIDIA, Université Libre de Bruxelles, Belgium. Published in European Journal of Operational Research López-Ibáñez and Stützle (2014).

López-Ibáñez, M., Stützle, T., 2014. Automatically improving the anytime behaviour of optimisation algorithms. European Journal of Operational Research 235, 569–582. doi:10.1016/j.ejor.2013.10.043.

Marmion, M.E., Mascia, F., López-Ibáñez, M., Stützle, T., 2013. Automatic design of hybrid stochastic local search algorithms, in: Blesa, M.J., Blum, C., Festa, P., Roli, A., Sampels, M. (Eds.), Hybrid Metaheuristics. Springer, Heidelberg. volume 7919 of *Lecture Notes in Computer Science*, pp. 144–158. doi:10.1007/978-3-642-38516-2_12.

Martin, R.C., 1996a. The Liskov substitution principle. C++ Report 8, 14.

Martin, R.C., 1996b. The open-closed principle. More C++ gems 19, 9.

Martin, R.C., 2011. The clean coder: a code of conduct for professional programmers. Pearson Education.

Martín-Santamaría, R., Sánchez-Oro, J., Pérez-Peló, S., Duarte, A., 2022. Strategic oscillation for the balanced minimum sum-of-squares clustering problem. Information Sciences 585, 529–542. doi:10.1016/j.ins.2021.11.048.

Martín-Santamaría , R., López-Sánchez , A.D., Delgado-Jalón , M.L., Colmenar , J.M., 2021. An efficient algorithm for crowd logistics optimization. Mathematics 9. doi:10.3390/math9050509.

Mascia, F., López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., 2014. Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools. Computers & Operations Research 51, 190–199. doi:10.1016/j.cor.2014.05.020.

Neri, F., Cotta, C., Moscato, P. (Eds.), 2011. Handbook of Memetic Algorithms. volume 379 of *Studies in Computational Intelligence*. Springer.

Pagnozzi, F., Stützle, T., 2019. Automatic design of hybrid stochastic local search algorithms for permutation flowshop problems. European Journal of Operational Research 276, 409–421. doi:10.1016/j.ejor.2019.01.018.

Parr, T.J., Quong, R.W., 1995. ANTLR: A predicated-LL (k) parser generator. Software — Practice & Experience 25, 789–810.

Rasku, J., Musliu, N., Kärkkäinen, T., 2019. On automatic algorithm configuration of vehicle routing problem solvers. Journal on Vehicle Routing Algorithms 2, 1–22. doi:10.1007/s41604-019-00010-9.

Stützle, T., López-Ibáñez, M., 2019. Automated design of metaheuristic algorithms, in: Gendreau, M., Potvin, J.Y. (Eds.), Handbook of Metaheuristics. Springer. volume 272 of *International Series in Operations Research & Management Science*, pp. 541–579. doi:10.1007/978-3-319-91086-4_17.

Swan, J., Adriaensen, S., Barwell, A.D., Hammond, K., White, D.R., 2019. Extending the "open-closed principle" to automated algorithm configuration. Evolutionary Computation 27, 173–193. doi:10.1162/evco_a_00245.

Swan, J., Adriaensen, S., Brownlee, A.E.I., Hammond, K., Johnson, C.G., Kheiri, A., Krawiec, F., Merelo, J.J., Minku, L.L., Özcan, E., Pappa, G., García-Sánchez, P., Sörensen, K., Voß, S., Wagner, M., White, D.R., 2022. Metaheuristics "in the large". European Journal of Operational Research 297, 393–406. doi:10.1016/j.ejor.2021.05.042.

Xavier, A.E., Xavier, V.L., 2011. Solving the minimum sum-of-squares clustering problem by hyperbolic smoothing and partition into boundary and gravitational regions. Pattern Recognition 44, 70–77. doi:10.1016/j.patcog.2010.07.004.

Part III

Additional Publications during thesis development

Contents

Chapter 10

Journal articles indexed in JCR & SJR

Additional publications in journals indexed in JCR and LNCS. Publications are ordered by acceptance date, from oldest to more recent. Submitted publications not yet published are last in the list.

10.1 On the analysis of the influence of the evaluation metric in community detection over social networks

Title	On the analysis of the influence of the evaluation metric in community detection over social networks
Authors	Sergio Pérez-Peló, Jesús Sánchez-Oro, Raúl Martín-Santamaría and Abraham Duarte
Publication date	2019
Journal	Electronics
Publisher	MDPI
ISBN/ISSN	2079-9292
Impact Factor	2.412 (2019)
Rank by Impact Factor	125/266 (Q2, Engineering, Electrical & Electronic)
DOI	`https://doi.org/10.3390/electronics8010023`

10.2 Solving the regenerator location problem with an Iterated Greedy approach

Title	Solving the regenerator location problem with an Iterated Greedy approch
Authors	Juan David Quintana, Raúl Martín-Santamaría, Jesís Sánchez-Oro and Abraham Duarte
Publication date	2021
Journal	Applied Soft Computing
Publisher	Science Direct
ISBN/ISSN	1568-4946
Impact Factor	8.263 (2021)
Rank by Impact Factor	11/112 (Q1, Computer Science, Interdisciplinary Applications)
DOI	`https://doi.org/10.1016/j.asoc.2021.107659`

10.3 WebGE: An Open-Source Tool for Symbolic Regression Using Grammatical Evolution

Title	WebGE: An Open-Source Tool for Symbolic Regression Using Grammatical Evolution
Authors	José Manuel Colmenar Verdugo, Raúl Martín-Santamaría and José Ignacio Hidalgo
Publication date	2022
Journal	Lecture Notes in Computer Science
Publisher	Springer International Publishing
ISBN/ISSN	978-3-031-02462-7
SJR	0.41 (2021)
Rank in SJR	Q2, Computer Science (miscellaneous)
DOI	`https://doi.org/10.1007/978-3-031-02462-7_18`

10.4 A Scatter Search Approach for the Parallel Row Ordering Problem

Title	A Scatter Search Approach for the Parallel Row Ordering Problem
Authors	Raúl Martín-Santamaría, José Manuel Colmenar Verdugo and Abraham Duarte Muñoz
Publication date	2023
Journal	Lecture Notes in Computer Science
Publisher	Springer International Publishing
ISBN/ISSN	978-3-031-26504-4
SJR	0.41 (2021)
Rank in SJR	Q2, Computer Science (miscellaneous)
DOI	https://doi.org/10.1007/978-3-031-26504-4_40

Chapter 11

Research presented in international and national conferences

Research projects presented both in national and international conferences, ordered by conference celebration date, from oldest to more recent.

11.1 A meta-heuristic approach for the Vehicle Routing Problem with occasional drivers

Title	A meta-heuristic approach for the Vehicle Routing Problem with occasional drivers
Authors	Raúl Martín-Santamaría, Ana Dolores López-Sánchez, José Manuel Colmenar Verdugo and María Luisa Delgado Jalón
Conference	7th Workshop of the EURO Working Group on Vehicle Routing and Logistics optimization (VeRoLog2019)
Celebration date	02/06/2019
Location	Sevilla, España
Organizing entity	EURO Working group

11.2 Using the Optaplanner solver

Title	Using the Optaplanner solver
Authors	Raúl Martín Santamaría
Conference	7th Workshop of the EURO Working Group on Vehicle Routing and Logistics optimization (VeRoLog2019)
Celebration date	02/06/2019
Location	Sevilla, España
Organizing entity	EURO Working group

11.3 A Variable Neighborhood Search approach for the Maximum Quasi-clique Problem

Title	A Variable Neighborhood Search approach for the Maximum Quasi-clique Problem
Authors	Raúl Martín-Santamaría and José Manuel Colmenar Verdugo
Conference	8th International Conference on Variable Neighborhood Search (ICVNS2022)
Celebration date	22/03/2021
Location	Abu Dhabi, Emiratos Árabes Unidos
Organizing entity	Khalifa University

11.4 Un algoritmo eficiente para el problema de disposición de instalaciones en dos filas

Title	Un algoritmo eficiente para el problema de disposición de instalaciones en dos filas
Authors	Raúl Martín-Santamaría, Alberto Herrán González, José Manuel Colmenar Verdugo and Abraham Duarte Muñoz
Conference	XIV Spanish Congress on Metaheuristics, Evolutionary and Bioinspired Algorithms (MAEB2021)
Celebration date	22/09/2021
Location	Málaga, España
Organizing entity	Universidad de Málaga

11.5 MORK: Metaheuristic Optimization framewoRK

Title	MORK: Metaheuristic Optimization framewoRK
Authors	Raúl Martín-Santamaría, José Manuel Colmenar Verdugo and Abraham Duarte Muñoz
Conference	Doctoral Consortium del Congreso de la Asociación Española de Inteligencia Artificial (CAEPIA-DC 2021)
Celebration date	22/09/2021
Location	Málaga, España
Organizing entity	Universidad de Málaga

11.6 A Scatter Search approach for the Parallel Row Ordering Problem

Title	A Scatter Search approach for the Parallel Row Ordering Problem
Authors	Raúl Martín-Santamaría, José Manuel Colmenar Verdugo and Abraham Duarte Muñoz
Conference	14th Metaheuristics International Conference (MIC2022)
Celebration date	11/07/2022
Location	Ortigia-Syracuse, Italy
Organizing entity	University of Catania

11.7 A VNS approach for the combined cell layout problem

Title	A VNS approach for the combined cell layout problem
Authors	Raúl Martín-Santamaría, José Manuel Colmenar Verdugo and Abraham Duarte Muñoz
Conference	9th International Conference on Variable Neighborhood Search (ICVNS2022)
Celebration date	25/10/2022
Location	Abu Dhabi, Emiratos Árabes Unidos
Organizing entity	Khalifa University

Part IV

Appendix

Appendix A

Resumen en castellano

De acuerdo al artículo 22 de la Normativa Reguladora de los Estudios de Doctorado de la Universidad Rey Juan Carlos, aprobada en Consejo de Gobierno de 07/06/2019, se provee un resumen en castellano del contenido completo de la tesis, incluyendo específicamente los antecedentes, objetivos, metodología, resultados y conclusiones obtenidas.

A.1 Introducción

Todos los días tomamos decenas de decisiones: qué ruta escoger para llegar al trabajo en función del tráfico, cómo organizar nuestros armarios, o qué productos comprar para preparar nuestras comidas son ejemplos de ello. Los problemas de optimización no se limitan a decisiones de nuestro día a día, sino que afectan a disciplinas tan diversas como las ingenierías, la economía o la logística. Todos los problemas de optimización tienen algo en común: maximizar o minimizar uno o varios objetivos, de acuerdo a una serie de restricciones.

Las técnicas para resolver problemas de optimización se pueden clasificar en dos grandes grupos: los métodos exactos y los métodos aproximados. Los primeros son capaces de encontrar la mejor solución posible a un problema dado, de acuerdo a sus restricciones, pero suelen requerir una gran potencia de cómputo, y suelen degradar su rendimiento en problemas reales donde el tamaño del problema es grande. Por otra parte, los métodos aproximados, como las heurísticas y las metaheurísticas, pueden encontrar soluciones de gran calidad utilizando pocos recursos computacionales, pero no pueden certificar si una solución dada es óptima, o si, por el contrario, existen soluciones mejores.

Mientras que las heurísticas y metaheurísticas se han vuelto unas de las técnicas más populares para resolver problemas de optimización, trabajos previos de la literatura han criticado la falta de un marco metodológico que cubra aspectos como la comparativa entre aproximaciones, la reproducibilidad experimental y la reusabilidad de las componentes algorítmicas implementadas por los investigadores [2, 3].

El resto del resumen está organizado como sigue. En la sección A.2, se presentan la hipótesis de partida y los objetivos propuestos. En la siguiente sección, A.3, se resumen los aspectos más relevantes de la propuesta metodológica, cuya implementación se validará en la sección A.4, utilizando tres problemas de optimización pertenecientes a familias completamente diferentes. Para finalizar, en la sección A.5 se presentan las conclusiones y posibles líneas de trabajo futuras.

A.2 Hipótesis y objetivos

Como se ha introducido previamente en la sección A.1, existe una falta de comprensión en la comunidad investigadora sobre la configuración automática de algoritmos y la selección de instancias de referencia. Por lo tanto, nuestra principal hipótesis es que proponiendo una metodología holística podemos mejorar la robustez, calidad y reproducibilidad de los enfoques metaheurísticos minimizando al mismo tiempo el esfuerzo para desarrollarlos.

El objetivo principal de esta tesis doctoral es desarrollar y validar una nueva metodología basada en evidencia científica para la aplicación de métodos metaheurísticos en problemas de optimización. En concreto, los objetivos específicos son los siguientes:

1. Proponer un enfoque metodológico para aumentar la reproducibilidad de los resultados empíricos, minimizando el número de decisiones que deben tomar los investigadores. En concreto, nos centraremos en tres puntos clave: la selección automática de instancias de prueba, en función de sus características; la generación y validación completamente automatizada de algoritmos; y, por último, la generación de artefactos para garantizar la reproducibilidad y favorecer la reusabilidad de las propuestas.

2. Desarrollar y publicar un conjunto de herramientas informáticas de código abierto accesibles públicamente a toda la comunidad científica, que implementen y automaticen la metodología propuesta.

3. Validar la metodología propuesta con problemas conocidos de optimización combinatoria de diferentes familias no relacionadas. Este objetivo incluye la publicación de todos los artefactos (código fuente, instancias, artefactos ejecutables y resultados procesados) en repositorios de acceso público.

4. Publicar todos los resultados en revistas relevantes y conferencias nacionales e internacionales.

A.3 Propuesta metodológica

En esta sección se resume la propuesta metodológica desarrollada en la tesis, cuya descripción completa puede encontrarse en el capítulo 2. En la figura A.1 se muestran los tres aspectos clave a tratar: la selección automática de instancias, la configuración automática de algoritmos y la generación de artefactos.

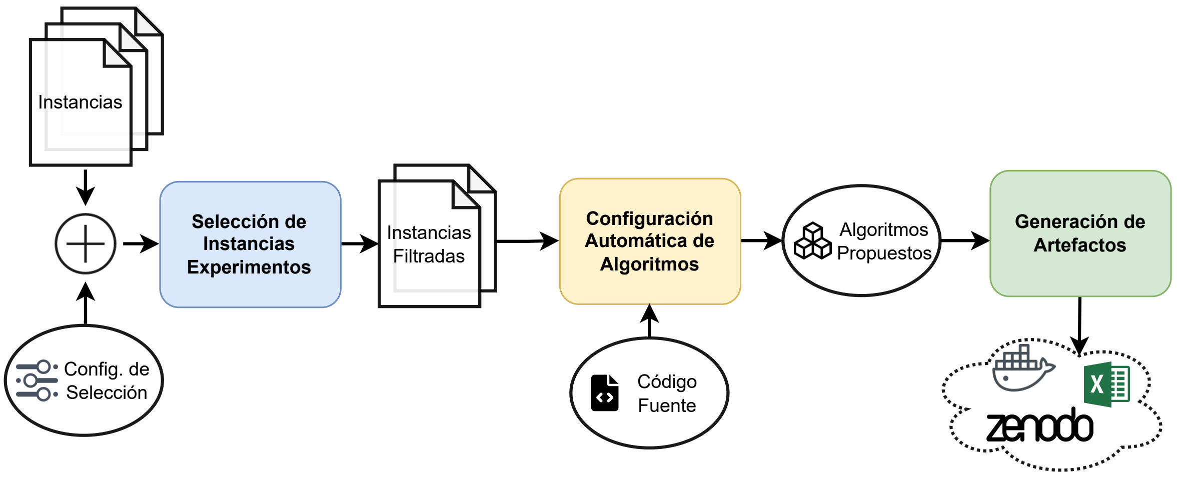


Figure A.1: Propuesta metodológica, representando con diferentes colores los tres aspectos claves de la propuesta: la selección automática de instancias, la configuración automática de algoritmos y la generación de artefactos.

La primera parte de la propuesta se centra en la selección automática de instancias de acuerdo a sus características. Para ello, es necesario que el investigador defina una lista de características a utilizar. Por ejemplo, en el caso de un grafo, posibles características podrían ser su densidad, el número de vértices y aristas, el número de componentes conexas, etc. Dada la lista de características, cuyo número puede ser tan grande como requiera el investigador, se propone el uso de métodos probados como Principal Component Analysis (PCA) y *clustering* basado en *k-means*, para determinar qué combinación de características son relevantes, agrupando así instancias similares. De esta forma, es posible elegir un subconjunto de instancias representativo para realizar la experimentación posterior.

La segunda parte propone un esquema de generación automática de algoritmos y su posterior validación de forma completamente automática, utilizando una aproximación basada en la generación de gramáticas a partir del código y las componentes algorítmicas propuestas por el usuario. La propuesta consiste en analizar el código del usuario en tiempo de ejecución, detectando todas las componentes algorítmicas disponibles, para entender qué dependencias existen entre todas ellas, y generar un árbol con todos los posibles algoritmos que podrían ser generados, filtrando todas aquellas combinaciones que no sean válidas. Este árbol puede ser transformado al espacio de parámetros utilizado por diferentes herramientas de optimización de parámetros, como `irace`, que tratarán de encontrar la mejor combinación de los mismos. Para poder evaluar el rendimiento de los algoritmos en diferentes momentos del tiempo, proponemos usar la métrica Area Under Curve (AUC), es decir, evaluar el rendimiento de cada algoritmo no en momentos arbitrarios de tiempo de ejecución, sino durante la evolución completa de la función objetivo.

Por último, la tercera parte de la propuesta trata de aspectos como el estandarizado de los artefactos ejecutables. En este sentido se propone utilizar contenedores para garantizar que la propuesta pueda ser ejecutada fácilmente en el futuro usando plataformas como Code Ocean [1]. Además, se listan los artefactos que deben ser publicados de forma obligatoria, incluyendo el código fuente completo de la propuesta, el conjunto de instancias original, y todos aquellos scripts y resultados intermedios que se hayan utilizado para generar los resultados de las publicaciones. Todos estos artefactos pueden ser publicados en plataformas como Zenodo [2].

La propuesta no se limita a ser puramente teórica, y su funcionamiento se demuestra en la siguiente sección, utilizando tres problemas de optimización pertenecientes a familias diferentes: un problema de *clustering*, un problema de rutas y un problema de disposición de instalaciones. La implementación que soporta esta metodología, denominada *Mork*, del inglés *Metaheuristic Optimization framewoRK*, se encuentra disponible de forma pública en GitHub [3].

A.4 Resultados

El primer problema elegido para demostrar la propuesta es el Vehicle Routing Problem with Ocassional Drivers (VRPOD), llamado en castellano problema de optimización de rutas con conductores ocasionales. La familia de problemas de optimización de rutas son un clásico en la literatura, cuyo objetivo suele ser una variante del siguiente: dado un almacén central que contiene los pedidos a repartir a un conjunto de clientes, se busca crear las rutas necesarias para satisfacer la demanda, minimizando el coste incurrido por la distancia recorrida. En esta variante del problema se dispone, además, de conductores ocasionales, que podrían ser hipotéticos clientes que han acudido a comprar al almacén, y que, si se les compensa de forma adecuada, podrían acceder a repartir uno de los paquetes, siempre que el destino del paquete sea cercano o esté en la ruta al destino final del cliente.

El segundo problema es el Balanced Minimum Sum-of-Squares Clustering (BMSSC), que consiste en agrupar un conjunto de elementos en grupos o *clusters* del mismo tamaño, de forma que elementos con características similares pertenezcan al mismo clúster. La función de similitud utilizada es la suma de distancias cuadradas (*sum-of-squares*, en Inglés) Entre las aplicaciones prácticas cabe destacar la minería de datos, reconocimiento de patrones y procesamiento de imágenes.

El último problema, llamado Space-Free Double Row Facility Layout Problem (SF-DRFLP), consiste en organizar un conjunto de elementos arbitrario, llamados *facilities*, en una rejilla formada por dos filas, de forma que se minimice coste de trasladar elementos entre cada par de *facilities*. Un ejemplo de aplicación podría ser la distribución de servicios en salas de un recinto, de forma que, sabiendo el número de personas que necesita cada servicio, o la secuencia de servicios que se suelen utilizar, se

[1]`https://codeocean.com/`
[2]`https://zenodo.org/`
[3]`https://github.com/mork-optimization/mork`

minimice la suma de los desplazamientos totales entre cada par de *facilities.* A diferencia de otras variantes de la familia de problemas, en este no se permiten espacios libres entre *facilities*, por lo que todas deben ser consecutivas y el comienzo de ambas filas debe estar alineado.

En la tabla A.1, se presenta el resumen de los resultados comparando las aproximaciones originales, etiquetadas como *Trabajo previo*, contra las aproximaciones generadas utilizando la metodología explicada en la sección A.3, etiquetadas como *Config. Automática.* Para cada problema, se muestran las siguientes métricas: número de veces que la configuración generada alcanza el mejor valor conocido para todas las instancias (#Times reaches *bkv*); número de veces que la configuración generada alcanza el mejor valor al menos una vez (#Instances finds *bkv*); promedio de desviación porcentual del mejor valor encontrado por una configuración respecto al mejor valor conocido (%Dev Min); y finalmente, promedio de desviación porcentual para todas las iteraciones al mejor valor conocido (%Dev Avg).

SF-DRFLP	Config. Automática	Trabajo previo
#Times reaches *bkv*	485 (19.96%)	895 (36.83%)
#Instances finds *bkv*	78 (96.30%)	42 (51.85%)
%Dev Min	0.00	0.01
%Dev Avg	0.45	0.03
BMSSC		
#Times reaches *bkv*	221 (29.47%)	206 (27.47%)
#Instances finds *bkv*	18 (72.00%)	17 (68.00%)
%Dev Min	0.03	0.20
%Dev Avg	0.46	0.46
VRPOD		
#Times reaches *bkv*	3816 (30.29%)	2373 (18.83%)
#Instances finds *bkv*	372 (88.57%)	132 (31.43%)
%Dev Min	1.55	0.68
%Dev Avg	5.06	1.91

Table A.1: Comparación entre las configuraciones generadas automáticamente para cada uno de los problemas y las aproximaciones originales. *bkv* son las iniciales en inglés de *best known value*, es decir, mejor valor conocido.

Como se puede observar, las configuraciones generadas de forma automática encuentran el mejor valor para la gran mayoría de instancias, mejorando o igualando los resultados de la propuesta previa basada en configuración manual de los experimentos, para los tres problemas propuestos, todo ello utilizando un procedimiento reproducible y automático. Para un análisis completo, ver el capítulo 4.

A.5 Conclusiones y trabajos futuros

En esta tesis se ha propuesto una nueva metodología para desarrollar aproximaciones heurísticas y metaheurísticas para diferentes clases de problemas de optimización, teniendo en cuenta los problemas más comunes existentes, especialmente aquellos relacionados con la reproducibilidad experimental y automatización de decisiones.

Como se ha demostrado en la sección A.4, hemos verificado la hipótesis inicial, es decir, es posible generar configuraciones de forma totalmente automática cuyo rendimiento iguale o incluso mejore aproximaciones configuradas de forma manual. Para ello, se han utilizado tres problemas pertenecientes a familias diferentes, utilizando estructuras para representar la solución y sus respectivos movimientos completamente dispares, mejorando en todos ellos el estado del arte existente.

Una limitación de la metodología es que no se puede garantizar la optimalidad de las soluciones obtenidas por los algoritmos generados, es decir, pueden existir diferentes configuraciones de parámetros que generen mejores resultados que las configuraciones propuestas. Sin embargo, nuestro objetivo principal no es garantizar dicha optimalidad, sino asegurarnos de que podemos construir, ejecutar y validar el rendimiento de las aproximaciones, de forma que garanticemos la reproducibilidad experimental, todo de forma procedural, igualando o mejorando los resultados existentes en el estado del arte, especialmente aquellos que utilizan aproximaciones basadas en configuración manual.

A lo largo de este proyecto de investigación, que culmina con la presente tesis, se han publicado 5 artículos JCR, cuatro de ellos en revistas Q1 del área, y el restante en una revista Q2, además de haber realizado múltiples contribuciones a congresos tanto nacionales como internacionales. Todos los artefactos generados a partir de la investigación realizada en esta tesis se encuentran disponibles de forma pública en Zenodo. Ver sección 5.1 para más detalle.

Como trabajos futuros, varias líneas interesantes son las siguientes: estudiar la compatibilidad con otros motores de optimización de parámetros como alternativa a `irace`; expandir la lista de componentes disponible listos para usar, añadiendo metaheurísticas basadas en poblaciones; estudiar la aplicación de esta metodología a problemas multi-objetivo; y, por último, finalizar la implementación basada en sistemas distribuidos para permitir escalabilidad horizontal, que puede ser fácilmente aprovechable en entornos de cómputo modernos.

Bibliography

[1] A. R. Amaral, “A heuristic approach for the double row layout problem,” *Annals of Operations Research*, pp. 1–36, 2020.

[2] T. Bartz-Beielstein, C. Doerr, D. v. d. Berg, J. Bossek, S. Chandrasekaran, T. Eftimov, A. Fischbach, P. Kerschke, W. La Cava, M. Lopez-Ibanez, *et al.*, “Benchmarking in optimization: Best practice and open issues,” *arXiv preprint arXiv:2007.03488*, 2020.

[3] G. Kendall, R. Bai, J. Błazewicz, P. De Causmaecker, M. Gendreau, R. John, J. Li, B. McCollum, E. Pesch, R. Qu, *et al.*, “Good laboratory practice for optimization research,” *Journal of the Operational Research Society*, vol. 67, no. 4, pp. 676–689, 2016.

[4] R. E. Ladner, “On the Structure of Polynomial Time Reducibility,” *Journal of the ACM*, vol. 22, pp. 155–171, Jan. 1975.

[5] L. A. Hemaspaandra, “Sigact news complexity theory column 36,” *ACM SIGACT News*, vol. 33, no. 2, pp. 34–47, 2002.

[6] A. Wigderson, “P, NP and mathematics – a computational complexity perspective,” in *Proceedings of the International Congress of Mathematicians Madrid, August 22–30, 2006* (M. Sanz-Solé, J. Soria, J. L. Varona, and J. Verdera, eds.), pp. 665–712, Zuerich, Switzerland: European Mathematical Society Publishing House, May 2007.

[7] G. B. Dantzig, A. Orden, P. Wolfe, *et al.*, “The generalized simplex method for minimizing a linear form under linear inequality restraints,” *Pacific Journal of Mathematics*, vol. 5, no. 2, pp. 183–195, 1955.

[8] J. E. Mitchell, “Branch-and-cut algorithms for combinatorial optimization problems,” *Handbook of applied optimization*, vol. 1, no. 1, pp. 65–77, 2002.

[9] M. Gendreau, J.-Y. Potvin, *et al.*, *Handbook of metaheuristics*, vol. 2. Springer, 2010.

[10] F. W. Glover and G. A. Kochenberger, *Handbook of metaheuristics*, vol. 57. Springer Science & Business Media, 2006.

[11] M. Birattari, “F-race for tuning metaheuristics,” in *Tuning metaheuristics*, pp. 85–115, Springer, 2009.

[12] A. Corominas, A. García-Villoria, and R. Pastor, “a systematic procedure based on calibra and the nelder & mead algorithm for fine-tuning metaheuristics,” *Journal of the Operational Research Society*, vol. 64, no. 2, pp. 276–282, 2013.

[13] T. Stützle and M. López-Ibáñez, “Automated design of metaheuristic algorithms,” in *Handbook of metaheuristics*, pp. 541–579, Springer, 2019.

[14] B. Adenso-Diaz and M. Laguna, “Fine-tuning of algorithms using fractional experimental designs and local search,” *Operations Research*, vol. 54, pp. 99–114, Jan. 2006.

[15] M. Birattari, T. Stützle, L. Paquete, K. Varrentrapp, *et al.*, “A racing algorithm for configuring metaheuristics.,” in *Gecco*, vol. 2 of *GECCO'02*, p. 11–18, 2002.

[16] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, “Paramils: an automatic algorithm configuration framework,” *Journal of Artificial Intelligence Research*, vol. 36, pp. 267–306, 2009.

[17] M.-C. Riff and E. Montero, “A new algorithm for reducing metaheuristic design effort,” in *2013 IEEE Congress on Evolutionary Computation*, pp. 3283–3290, IEEE, 2013.

[18] F. Hutter, H. H. Hoos, and K. Leyton-Brown, “Sequential model-based optimization for general algorithm configuration,” in *International conference on learning and intelligent optimization*, pp. 507–523, Springer, 2011.

[19] O. Maron and A. W. Moore, “The racing algorithm: Model selection for lazy learners,” *Artificial Intelligence Review*, vol. 11, no. 1, pp. 193–225, 1997.

[20] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle, “F-race and iterated f-race: An overview,” *Experimental methods for the analysis of optimization algorithms*, pp. 311–336, 2010.

[21] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, T. Stützle, and M. Birattari, “The irace package: Iterated racing for automatic algorithm configuration,” *Operations Research Perspectives*, vol. 3, pp. 43–58, 2016.

[22] M. De Souza, M. Ritt, M. López-Ibáñez, and L. Pérez Cáceres, “Acviz: A tool for the visual analysis of the configuration of algorithms with irace,” *Operations Research Perspectives*, vol. 8, p. 100186, 2021.

[23] M. López-Ibáñez, J. Branke, and L. Paquete, “Reproducibility in evolutionary computation,” *ACM Transactions on Evolutionary Learning and Optimization*, vol. 1, no. 4, pp. 1–21, 2021.

[24] M. Birattari, M. Zlochin, and M. Dorigo, “Towards a theory of practice in metaheuristics design: A machine learning perspective,” *RAIRO-Theoretical Informatics and Applications*, vol. 40, no. 2, pp. 353–369, 2006.

[25] J. A. Olvera-López, J. A. Carrasco-Ochoa, J. F. Martínez-Trinidad, and J. Kittler, "A review of instance selection methods," *Artificial Intelligence Review*, vol. 34, no. 2, pp. 133–143, 2010.

[26] E.-G. Talbi, "Machine learning into metaheuristics: A survey and taxonomy," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–32, 2021.

[27] I. Jolliffe, "Principal component analysis," *Encyclopedia of statistics in behavioral science*, 2005.

[28] I. T. Jolliffe and J. Cadima, "Principal component analysis: a review and recent developments," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20150202, 2016.

[29] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1,14, pp. 281–297, Oakland, CA, USA, 1967.

[30] V. Satopaa, J. Albrecht, D. Irwin, and B. Raghavan, "Finding a" kneedle" in a haystack: Detecting knee points in system behavior," in *2011 31st international conference on distributed computing systems workshops*, pp. 166–171, IEEE, 2011.

[31] J. Swan, S. Adriaensen, C. Johnson, A. Kheiri, F. Krawiec, J. Merelo Guervós, L. Minku, E. Özcan, G. Pappa, P. García-Sánchez, K. Sörensen, S. Voss, M. Wagner, and D. White, "Metaheuristics "in the large"," *European Journal of Operational Research*, vol. 297, pp. 393–406, 03 2022.

[32] R. C. Martin, "The Liskov substitution principle," *C++ Report*, vol. 8, no. 3, p. 14, 1996.

[33] R. C. Martin, "The open-closed principle," *More C++ gems*, vol. 19, no. 96, p. 9, 1996.

[34] J. Swan, S. Adriænsen, A. D. Barwell, K. Hammond, and D. R. White, "Extending the "Open-Closed Principle" to Automated Algorithm Configuration," *Evolutionary Computation*, vol. 27, pp. 173–193, Mar. 2019.

[35] A. R. KhudaBukhsh, L. Xu, H. H. Hoos, and K. Leyton-Brown, "Satenstein: Automatically building local search sat solvers from components," *Artificial Intelligence*, vol. 232, pp. 20–42, 2016.

[36] F. Pagnozzi and T. Stützle, "Automatic design of hybrid stochastic local search algorithms for permutation flowshop problems," *European Journal of Operational Research*, vol. 276, no. 2, pp. 409–421, 2019.

[37] F. Mascia, M. López-Ibáñez, J. Dubois-Lacoste, and T. Stützle, "Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools," *Computers & Operations Research*, vol. 51, pp. 190–199, 2014.

[38] J. Dreo, A. Liefooghe, S. Verel, M. Schoenauer, J. J. Merelo, A. Quemy, B. Bouvier, and J. Gmys, "Paradiseo: From a modular framework for evolutionary

computation to the automated design of metaheuristics: 22 years of paradiseo," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '21, (New York, NY, USA), p. 1522–1530, Association for Computing Machinery, 2021.

[39] F. Mascia, M. López-Ibáñez, J. Dubois-Lacoste, and T. Stützle, "From grammars to parameters: Automatic iterated greedy design for the permutation flowshop problem with weighted tardiness," in *Learning and Intelligent Optimization* (G. Nicosia and P. Pardalos, eds.), pp. 321–334, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[40] J. Rasku, N. Musliu, and T. Kärkkäinen, "On automatic algorithm configuration of vehicle routing problem solvers," *Journal on Vehicle Routing Algorithms*, vol. 2, no. 1, pp. 1–22, 2019.

[41] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[42] ACM, "Artifact review and badging, version 2.0," 2021.

[43] A. Clyburne-Sherin, X. Fei, and S. A. Green, "Computational reproducibility via containers in psychology," *Meta-psychology*, vol. 3, 2019.

[44] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & operations research*, vol. 13, no. 5, pp. 533–549, 1986.

[45] I. Boussaïd, J. Lepagnot, and P. Siarry, "A survey on optimization metaheuristics," *Information Sciences*, vol. 237, pp. 82–117, July 2013.

[46] T. Feo and M. Resende, "A probabilistic heuristic for a computationally difficult set covering problem," *Operations Research Letters*, vol. 8, pp. 67–71, 1989.

[47] T. A. Feo, M. G. C. Resende, and S. H. Smith, "A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set," *Operations Research*, vol. 42, no. 5, pp. 860–878, 1994.

[48] R. Martí, A. Martínez-Gavara, J. Sánchez-Oro, and A. Duarte, "Tabu search for the dynamic Bipartite Drawing Problem.," *Computers & OR*, vol. 91, pp. 1–12, 2018.

[49] A. Duarte, J. Sánchez-Oro, M. G. C. Resende, F. Glover, and R. Martí, "Greedy randomized adaptive search procedure with exterior path relinking for differential dispersion minimization.," *Inf. Sci.*, vol. 296, pp. 46–60, 2015.

[50] F. Glover, "Multi-start and strategic oscillation methods–principles to exploit adaptive memory," *Computing tools for modeling, optimization and simulation: interfaces in computer science and operations research*, pp. 1–24, 2000.

[51] M. Sevaux, A. Rossi, M. Soto, A. Duarte, and R. Martí, "GRASP with ejection chains for the dynamic memory allocation in embedded systems.," *Soft Comput.*, vol. 18, no. 8, pp. 1515–1527, 2014.

[52] K. Jajuga, A. Sokolowski, and H.-H. Bock, *Classification, clustering, and data analysis: recent advances and applications.* Springer Science & Business Media, 2012.

[53] R. Xu and D. Wunsch, "Survey of clustering algorithms," *IEEE Transactions on neural networks*, vol. 16, no. 3, pp. 645–678, 2005.

[54] A. E. Xavier and V. L. Xavier, "Solving the minimum sum-of-squares clustering problem by hyperbolic smoothing and partition into boundary and gravitational regions.," *Pattern Recognition*, vol. 44, no. 1, pp. 70–77, 2011.

[55] M. Mahajan, P. Nimbhorkar, and K. Varadarajan, "The Planar k-Means Problem is NP-Hard," in *WALCOM: Algorithms and Computation* (S. Das and R. Uehara, eds.), (Berlin, Heidelberg), pp. 274–285, Springer Berlin Heidelberg, 2009.

[56] D. J. Aloise, D. Aloise, C. T. M. Rocha, C. Ribeiro, J. R. Filho, and L. S. Moura, "Scheduling workover rigs for onshore oil production.," *Discrete Applied Mathematics*, vol. 154, pp. 695–702, Sept. 2006.

[57] A. Pyatkin, D. Aloise, and N. Mladenović, "NP-Hardness of balanced minimum sum-of-squares clustering.," *Pattern Recognition Letters*, vol. 97, pp. 44–45, 2017.

[58] T. F. González, "On the computational complexity of clustering and related problems," in *System Modeling and Optimization* (R. F. Drenick and F. Kozin, eds.), (Berlin, Heidelberg), pp. 174–182, Springer Berlin Heidelberg, 1982.

[59] E. Queiroga, A. Subramanian, and L. dos Anjos F. Cabral, "Continuous greedy randomized adaptive search procedure for data clustering," *Applied Soft Computing*, vol. 72, pp. 43–55, 2018.

[60] S. Chakraborty, D. Paul, and S. Das, "Hierarchical clustering with optimal transport," *Statistics & Probability Letters*, vol. 163, p. 108781, 2020.

[61] S. M. Mohammed, K. Jacksi, and S. Zeebaree, "A state-of-the-art survey on semantic similarity for document clustering using glove and density-based algorithms," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 22, no. 1, pp. 552–562, 2021.

[62] A. Hinneburg and D. A. Keim, "Optimal grid-clustering : Towards breaking the curse of dimensionality in high-dimensional clustering," in *Proceedings of the 25 th International Conference on Very Large Databases, 1999*, pp. 506–517, 1999.

[63] K. Wong, "A Short Survey on Data Clustering Algorithms," in *2015 Second International Conference on Soft Computing and Machine Intelligence (ISCMI)*, pp. 64–68, 2015.

[64] H. Steinhaus, "Sur la division des corps matériels en parties," *Bull. Acad. Polon. Sci. Cl. III.*, vol. 4, pp. 801–804, 1956.

[65] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: analysis and implementation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 881–892, 2002.

[66] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," tech. rep., Stanford, 2006.

[67] J. Xu and K. Lange, "Power k-means clustering," in *International Conference on Machine Learning*, pp. 6921–6931, PMLR, 2019.

[68] S. Chakraborty, D. Paul, S. Das, and J. Xu, "Entropy weighted power k-means clustering," in *International Conference on Artificial Intelligence and Statistics*, pp. 691–701, PMLR, 2020.

[69] L. R. Costa, D. Aloise, and N. Mladenović, "Less is more: basic variable neighborhood search heuristic for balanced minimum sum-of-squares clustering.," *Inf. Sci.*, vol. 415, pp. 247–253, 2017.

[70] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[71] R. Elshaer and H. Awad, "A taxonomic review of metaheuristic algorithms for solving the vehicle routing problem and its variants," *Computers & Industrial Engineering*, vol. 140, p. 106242, Feb. 2020.

[72] J. Allen, M. Piecyk, M. Piotrowska, F. McLeod, T. Cherrett, K. Ghali, T. Nguyen, T. Bektas, O. Bates, A. Friday, *et al.*, "Understanding the impact of e-commerce on last-mile light goods vehicle activity in urban areas: The case of london," *Transportation Research Part D: Transport and Environment*, vol. 61, pp. 325–338, 2018.

[73] R. Gevaers, E. Van de Voorde, T. Vanelslander, *et al.*, "Characteristics of innovations in last-mile logistics-using best practices, case studies and making the link with green and sustainable logistics," *Association for European Transport and contributors*, 2009.

[74] L. Ranieri, S. Digiesi, B. Silvestri, and M. Roccotelli, "A review of last mile logistics innovations in an externalities cost reduction vision," *Sustainability*, vol. 10, no. 3, p. 782, 2018.

[75] M. Janjevic and M. Winkenbach, "Characterizing urban last-mile distribution strategies in mature and emerging e-commerce markets," *Transportation Research Part A: Policy and Practice*, vol. 133, pp. 164–196, Mar. 2020.

[76] A. Sampaio, M. Savelsbergh, L. Veelenturf, and T. Van Woensel, "Chapter 15: Crowd-Based City Logistics," in *Sustainable Transportation and Smart Logistics*

(J. Faulin, S. E. Grasman, A. A. Juan, and P. Hirsch, eds.), pp. 381–400, Elsevier, 2019.

[77] V. Carbone, A. Rouquet, and C. Roussat, “The rise of crowd logistics: A new way to co-create logistics value,” *Journal of Business Logistics*, vol. 38, no. 4, pp. 238–252, 2017.

[78] A. Devari, A. G. Nikolaev, and Q. He, “Crowdsourcing the last mile delivery of online orders by exploiting the social networks of retail store customers,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 105, pp. 105 – 122, 2017.

[79] R. Botsman, “Crowdshipping: using the crowd to transform delivery,” *AFR Boss Magazine*, no. September 12, 2014.

[80] X. Guo, Y. J. L. Jaramillo, J. Bloemhof-Ruwaard, and G. Claassen, “On integrating crowdsourced delivery in last-mile logistics: A simulation study to quantify its feasibility,” *Journal of Cleaner Production*, vol. 241, p. 118365, 2019.

[81] M. D. Simoni, E. Marcucci, V. Gatta, and C. G. Claudel, “Potential last-mile impacts of crowdshipping services: a simulation-based evaluation,” *Transportation*, pp. 1–22, 2019.

[82] C. Archetti, M. Savelsbergh, and M. G. Speranza, “The vehicle routing problem with occasional drivers,” *European Journal of Operational Research*, vol. 254, no. 2, pp. 472 – 480, 2016.

[83] R. Martín-Santamaría, A. D. López-Sánchez, M. L. Delgado-Jalón, and J. M. Colmenar, “An Efficient Algorithm for Crowd Logistics Optimization,” *Mathematics*, vol. 9, p. 509, Mar. 2021.

[84] R. Satheesh Kumar, P. Asokan, S. Kumanan, and B. Varma, “Scatter search algorithm for single row layout problem in fms,” *Advances in Production Engineering & Management*, vol. 3, no. 4, pp. 193–204, 2008.

[85] D. M. Simmons, “One-dimensional space allocation: an ordering algorithm,” *Operations Research*, vol. 17, no. 5, pp. 812–826, 1969.

[86] M. Rubio-Sánchez, M. Gallego, F. Gortázar, and A. Duarte, “Grasp with path relinking for the single row facility layout problem,” *Knowledge-Based Systems*, vol. 106, pp. 1–13, 2016.

[87] A. Herrán, J. M. Colmenar, and A. Duarte, “An efficient variable neighborhood search for the space-free multi-row facility layout problem,” *European Journal of Operational Research*, 2021.

[88] P. Hungerländer, K. Maier, V. Pachatz, and C. Truden, “Exact and heuristic approaches for a new circular layout problem,” *SN Applied Sciences*, vol. 2, no. 6, pp. 1–22, 2020.

[89] M. Dahlbeck, "A mixed-integer linear programming approach for the t-row and the multi-bay facility layout problem," *European Journal of Operational Research*, 2021.

[90] A. Drira, H. Pierreval, and S. Hajri-Gabouj, "Facility layout problems: A survey," *Annual Reviews in Control*, vol. 31, pp. 255–267, Jan. 2007.

[91] J. Chung and J. Tanchoco, "The double row layout problem," *International Journal of Production Research*, vol. 48, no. 3, pp. 709–727, 2010.

[92] A. R. Amaral, "Optimal solutions for the double row layout problem," *Optimization Letters*, vol. 7, no. 2, pp. 407–413, 2013.

[93] L. D. Secchin and A. R. S. Amaral, "An improved mixed-integer programming model for the double row layout of facilities," *Optimization Letters*, vol. 13, no. 1, pp. 193–199, 2019.

[94] J. Chae and A. C. Regan, "A mixed integer programming model for a double row layout problem," *Computers & Industrial Engineering*, vol. 140, p. 106244, 2020.

[95] A. R. Amaral, "A mixed-integer programming formulation of the double row layout problem based on a linear extension of a partial order," *Optimization Letters*, vol. 15, no. 4, pp. 1407–1423, 2021.

[96] F. Glover and J.-K. Hao, "The case for strategic oscillation," *Annals of Operations Research*, vol. 183, no. 1, pp. 163–173, 2011.

[97] M. I. Malinen and P. Fränti, "Balanced k-means for clustering," in *Structural, Syntactic, and Statistical Pattern Recognition* (P. Fränti, G. Brown, M. Loog, F. Escolano, and M. Pelillo, eds.), (Berlin, Heidelberg), pp. 32–41, Springer Berlin Heidelberg, 2014.

[98] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE transactions on pattern analysis and machine intelligence*, no. 2, pp. 224–227, 1979.

[99] R. M. Aiex, M. G. Resende, and C. C. Ribeiro, "Ttt plots: a perl program to create time-to-target plots," *Optimization Letters*, vol. 1, no. 4, pp. 355–366, 2007.

[100] H. R. Lourenço, O. C. Martin, and T. Stützle, *Iterated Local Search*, pp. 320–353. Boston, MA: Springer US, 2003.

[101] A. Herrán, J. M. Colmenar, R. Martí, and A. Duarte, "A parallel variable neighborhood search approach for the obnoxious p-median problem," *International Transactions in Operational Research*, vol. 27, no. 1, pp. 336–360, 2020.

[102] J. Faulin and A. A. Juan, "The ALGACEA-1 method for the capacitated vehicle routing problem," *International Transactions in Operational Research*, vol. 15, no. 5, pp. 599–621, 2008.

[103] G. M. Buxey, “The vehicle scheduling problem and monte carlo simulation,” *Journal of the Operational Research Society*, vol. 30, no. 6, pp. 563–573, 1979.

[104] L. W. Jacobs and M. J. Brusco, “Note: A local-search heuristic for large set-covering problems,” *Naval Research Logistics (NRL)*, vol. 42, no. 7, pp. 1129–1140, 1995.

[105] R. Ruiz and T. Stützle, “A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem,” *European journal of operational research*, vol. 177, no. 3, pp. 2033–2049, 2007.

[106] M. Lozano, D. Molina, and C. García-Martínez, “Iterated greedy for the maximum diversity problem,” *European Journal of Operational Research*, vol. 214, no. 1, pp. 31–38, 2011.

[107] R. Ruiz, Q.-K. Pan, and B. Naderi, “Iterated greedy methods for the distributed permutation flowshop scheduling problem,” *Omega*, vol. 83, pp. 213–222, 2019.

[108] J. D. Quintana, R. Martin-Santamaria, J. Sanchez-Oro, and A. Duarte, “Solving the regenerator location problem with an iterated greedy approach,” *Applied Soft Computing*, vol. 111, p. 107659, 2021.

[109] M. López-Ibánez and T. Stützle, “Automatically improving the anytime behaviour of optimisation algorithms,” *European Journal of Operational Research*, vol. 235, no. 3, pp. 569–582, 2014.

[110] S. Cavero, E. G. Pardo, and A. Duarte, “Efficient iterated greedy for the two-dimensional bandwidth minimization problem,” *European Journal of Operational Research*, vol. 306, no. 3, pp. 1126–1139, 2023.

[111] I. Lozano-Osorio, J. Sánchez-Oro, A. Martínez-Gavara, A. D. López-Sánchez, and A. Duarte, “An efficient fixed set search for the covering location with interconnected facilities problem,” in *Metaheuristics* (L. Di Gaspero, P. Festa, A. Nakib, and M. Pavone, eds.), (Cham), pp. 485–490, Springer International Publishing, 2023.

[112] J. Mockus, “The Bayesian Approach to Local Optimization,” *Bayesian Approach to Global Optimization*, pp. 125–156, 1989.

[113] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization.,” *Journal of machine learning research*, vol. 13, no. 2, 2012.